Challenges for the Application of Migratory User Interfaces in Industrial Process Visualizations

Lukas Baron(✉) and Annerose Braune(✉)

Institute of Automation, Technische Universität Dresden, Dresden, Germany
{lukas.baron,annerose.braune}@tu-dresden.de

Abstract. The increasing familiarity of users with modern human-machine interaction concepts (e. g. touch gestures) and corresponding devices makes these devices and concepts interesting for industrial applications. However, the *combination* of well-established stationary devices and newer mobile devices may cause negative effects on human operator's workflows if the applications – industrial process visualizations in our case – are not well-designed to conform to the user's expectations. These expectations especially focus on the ability of simultaneously used devices to allow users to collaborate with each other or to change their devices frequently. This motivates the use of migratory user interfaces (MUI) which are able to change devices without losing relevant information, and thus, without interrupting the workflow. Hence, in this paper, we present excerpts of the established concept of MUIs and analyze it with respect to the demands given by the domain of industrial process visualizations. We are able to show that these demands require certain extensions of current MUI techniques, for example, explicit markup telling relevant and non-relevant parts of the UI state apart. Our review of the related work reveals that there is no suitable solution which meets the demands. In order to demonstrate the feasibility of migratory UIs in the industrial domain, we present a case study which focuses on extending already existing user interfaces with the required functionality for migration.

1 Introduction

New human-machine interaction concepts that make use of touch displays, motion sensing, etc. have come to mainstream popularity. This fosters the introduction, e. g. of mobile devices, in industrial applications. In such environments, user interfaces (UI) – process visualizations in our case – currently are designed specifically for a certain set of devices. Each device and its respective UI thereby has its own dedicated purpose which is not intended to change over its lifetime. Examples for such UIs are process visualizations in operator stations or on in-field devices like panel PCs. Mobile devices shall *supplement* such stationary devices in order to allow the process supervision and control to be handled more flexible. For example, a mobile device may allow access to certain process data from any location within the plant in order to conduct maintenance tasks. Such

M. Kurosu (Ed.): HCI 2016, Part II, LNCS 9732, pp. 364–378, 2016.
DOI: 10.1007/978-3-319-39516-6_35

tasks often require multiple users to cooperate with each other while being situated in different locations, e. g., one user is close to the device to maintain and another user keeps observing the process from the operating room. In such a scenario, the UI is expected to support collaborative functionalities. Moreover, users may need or want to change the currently used device during the conduction of a single task, for example, caused by changing their location away from a stationary device. However, doing this could have a negative effect on the user's effectiveness because it takes time to get the UI on the new device to a state from where the interaction – and the conduction of the task – can be continued. Briefly speaking, the user may need to enter data or navigate to the same part of the visualization on the new device.

Instead, such a change of device is supposed to work without interrupting the user's ongoing work – and thus, not reducing his/her effectiveness. By automatically taking into account the recent interaction history with the UI, the time needed to change the device and to proceed interaction can be reduced. Moreover, the same mechanism which allows the smooth change of the device can be used to realize collaborative UIs. This kind of freely transferable UIs – referred to as migratory or nomadic UIs (MUI) [1] – are state of the art in multimedia applications [9], home automation [4], or collaborative education software [8] but not yet in industrial environments.

Migration-related procedures may enable or at least support a more effective handling of tasks in which users may switch from stationary to mobile devices, e. g., to conduct maintenance or commissioning tasks. However, safety, security, and reliability are critical measures of quality for applications in the industrial domain, and thus, for user interfaces as well. Hence, special attention has to be payed to the satisfaction of these demands when realizing migratory UIs for industrial applications.

In this paper, we will briefly introduce the established concept of migratory UIs. Furthermore, the influence of the industrial domain on the functional parameters of MUIs will be discussed in detail. Finally, a case study is presented in order to show the feasibility in accordance with the requirements and to emphasize an industrial use case for migratory UIs.

2 Migratory User Interfaces

Before the industrial requirements for migratory user interfaces can be deduced, we want to introduce the MUI theory. In the first subsection the general characteristics are presented which are common to MUIs. The second section provides an excerpt of properties that allow the classification of migratory applications.

2.1 Functional Aspects

According to [3], an MUI is characterized by a transfer from a source device on which the user started interaction to a known target device on which to proceed immediately. The authors claim that the most important part is to transfer the UI *state* instead of exclusively transferring the static artifacts defining the UI's software implementation. The UI state is constituted by the complete user interaction history prior to the moment of migration, consisting of internally stored properties, the user's and the system's in- and output and called functions including the gained results. The objective of the state transfer is to achieve a *continuous interaction*. If either the target device differs in hard- or software or if its environment changes, usage continuity also requires adaption capabilities.

Aspects of distributed user interfaces (DUI) have to be considered if a UI concerning a specific technical process is split up into parts that are located on multiple devices [1]. For providing usage continuity, a migration can be applied in case of a change of the UI distribution, i. e., users swap devices, including an optional change of the device number or type. A change of distribution does not necessarily include a migration mechanism, though. Of course, such a redistribution implies a transfer of static UI elements to new devices. However, usage continuity relies on the state transfer. Hence, this applies even in scenarios where the UI distribution stays static.

In summary, the steps necessary for migrating a UI can be derived from the following three aspects of MUIs [1]:

1. **Distribution**: transfer of static UI components causing a change of the distribution configuration,
2. **State transfer**[1]: determination and transfer of the UI state which also needs to reflect the change of distribution,
3. **Adaption**[2]: static components of the UI have to be adapted to the changed device, environment, or execution runtime which may also require an adaption of the transferred state.

In [3], the term *migration engine* has been defined which covers the implementation realizing the migration process. Such an engine needs parameters in order to perform a migration, e. g., information about source and target devices, a set of concrete UI elements to migrate, the UI state to consider, the users associated with each device, etc. These migration parameters in combination with the migration engine determine what we call the *migratory behavior* which also includes the actions users have to take in order to provide the parameters. The migratory behavior provides *conformity to expectations* if the provided parameters lead to the intended result. This, however, requires a predictable operation of the migration engine.

2.2 MUI Classification

The (intended) behavior of an MUI can be described by means of 13 identified classifiers [1,3,10]. We will only present some selected ones which are relevant for this paper:

[1] In [1], this is referred to as the migratory aspect of a UI.
[2] also referred to as UI plasticity [1].

① **Initialization.** A migration can be triggered *automatically* or *on demand* by users. The trigger might originate from arbitrary devices connected to the UI system. If the triggering device is the migration source, it is called *pushing* – *pulling*, in case it is the target. To each single migration a direction can be assigned which distinguishes the triggering device from its passive counterpart. Thus a migration can be pushed from the triggering device as the source or pulled onto the triggering device as the target. Hybrid modes – pulling and pushing at the same time, e. g., when swapping content between devices – are also possible.

② **Scope.** The transfer of information may include the UI en bloc – *total* – or excerpts thereof – *partial*. In a redistribution scenario, the process may *distribute* the UI from one to many devices, *aggregate* from many to one device, or *mix* it with multiple devices as sources and targets as well.

③ **Adaption Type.** The UI adaption may be realized completely *dynamically* (at runtime), *precomputed* (at design time) by loading a complete *static* UI, or as an intermediate type, for instance, by using templates that are assembled at runtime during the migration procedure.

④ **System Architecture.** The system architecture describes the organization of the migration engine. The engine can either be distributed on each of the UI devices which arrange themselves autonomously – *peer-to-peer* – or concentrated on an additional entity acting as a migration *server*. In the latter case, target and source UIs act as migration *clients*. Furthermore, in such a migration-server and migration-client architectures, *client-based* and *server-based* approaches can be distinguished. This classification describes the entity responsible for determining and obtaining the migration parameters including the required UI state to transfer.

⑤ **Simultaneous Usability.** In multi-user and -device scenarios, many users may use a single device (N-1 relation), one user may use multiple devices (1-N), or many users use many devices (N-M), e. g., in a collaborative task. In the context of migratory and distributed applications, this term also includes the mode of state transfer which can be *continuous* (synchronizing different UIs over time) or *discrete* (transfer the state once, i. e., the state is outdated after the migration has been completed).

3 Industrial Demands

3.1 Software and Engineering Requirements

In order to classify the requirements of industrial process visualizations, we distinguish between a *design phase* and a *runtime* life cycle phase. The latter phase covers the operative period of the UI starting from the point in time when the UI has been connected to the technical process. Thus, this phase also includes timeframes in which the process itself is not functional but the UI is required to work (e. g. during maintenance or commissioning). Prior to this, the design phase covers planning, engineering, implementation, and testing of the UI.

An industrial visualization has to interact with complex and expensive technical systems which poses requirements to its reliability in terms of safety and security. Likewise, the accountability of the engineering results in specific requirements that need to be considered during the design phase. In particular, the UI's presentation and (migratory) behavior during runtime has to be predictable. The characteristics to be provided by UIs (e. g. visually) are specified in standards of the respective domain, company, or contractee. The functional features depend on the tasks that a user has to fulfill [6]. Hereby, the standards are amongst other things concerned with usability aspects which have an influence on presentational and behavioral parameters. Thus, the UI and potential migration capabilities have to ensure by design that users are not being disturbed, for example, through unneeded functions and UI elements or unexpected behavior.

3.2 UI Structure and Functionality

Industrial process visualizations usually consist of panels connected via navigation elements. However, primarily they contain elements for the display and manipulation of certain aspects or parts of the technical process [13]. Such elements are often associated with each other reflecting functional dependencies of their corresponding components within the process plant. This means that certain UI elements must not be divided arbitrarily (e. g. during partial migrations) but with respect to the user's task in a way that all needed functionality is available before and after a migration. Information about such relations are usually not explicitly available within visualization applications. For example, some pumps and valves are related to each other due to the pipes that connect them. But even if both, pipes and devices, are visibly connected in the visualization, the functional dependencies are not identifiable automatically. Such knowledge is provided only by domain experts, although not necessarily in a manner which is suitable for an automatic interpretation by machines. In terms of migration, this has to be considered in order to support the determination of optimal migration parameters for the users' workflow. If this knowledge is well formalized, some algorithms determining which elements belong together could be applied as proposed in [2]. However, the demand of predictable behavior (see Sect. 3.1) requires means for developers to amend or to override an algorithm's result by providing information manually. For that purpose, either the algorithm must be adjustable or its results have to be obtained during the design phase to be stored in an intermediate data structure that can be edited by the UI engineer in order to configure migration engines at runtime.

Another peculiarity of a process visualization is its real-time communication with process data servers or (non-real-time connections to) historical databases in order to provide a continuously updated process view (for example, visualized by changing and animating selected UI elements). Such process visualizations are extended by an authorization mechanism which is an important part of the security and safety design. It grants or denies read and write access to single data items with respect to the current UI device, its assigned user, as well as the user's roles within the company. Typically, there is only one user at a time with exclusive control rights to (a limited part of) the process – granted by a control token. During runtime, the tokens might be transferred, but only with mutual consent and granted access rights.

A UI's structural, functional, and perceptual design is strictly bound to its well-defined use case, the intended role of a user, the device, and the environment [14]. Thus, it is common that the same technical process is equipped with multiple individual (and individually designed) UIs. For example, consider the following scenario: An operator located in a control room with a stationary desktop PC wants to migrate some panels of a UI to service personnel in the field in order to give instructions. Both UIs have a completely different use case and thus a different level of detail. The operator's view is limited to the functionality necessary for nominal operations. On the other hand, servicing needs access to all parameters of the faulty device plus eventually further devices that are functionally associated or collocated. However, both UIs would not only differ in the displayed content, there may be organizational or presentational differences, too. That is why a migration engine would need to mediate between different use cases for corresponding UIs by adapting them in case of migration.

3.3 Implications for MUI Properties

As we explained in the previous section, the UIs considered in this contribution always need to communicate with process data servers. By transferring these data to the UI, they obviously become part of its state. Since this functionality can be presumed in any case, we have to separate different parts of the state from each other. In Fig. 1, the state is depicted decomposed into an injected and an internal state.

By *injected*, we refer to all information and process data that are imported exclusively from external providers and, thus, can be reconstructed without any additional information. The *internal* state is only known to the UI itself and may, for instance, be incorporated by internal variables which are created at runtime. Each part of the state can be used for output on the UI's presentational feature. Interactors of course intend to influence the internal state in order to provide user input. Algorithms f implemented within the UI (see Fig. 1) may calculate required information based on the injected and/or the internal state. In terms of state transfer, the injected state *can* be omitted (as it can be recreated on the target device) but in the industrial domain it *must* be omitted due to authorization mechanisms that may be bypassed otherwise. If bypassed, the migration engine may yield access to process data for users who are not supposed to have access. That is why we will concentrate on the internal state only.

Since the division of the state into the internal and the injected parts is usually only implemented implicitly within UIs, we need additional explicit markup that allows the automatic handling of the state transfer. Depending on the current situation, e. g., characterized by a certain task, in which a migration occurs, not the complete internal state is relevant. This results in a further division of the internal state into *relevant* and *non-relevant* parts of which only relevant parts will be migrated. In analogy with the definition of partial migration in Sect. 2.2, this can be considered as a *partial internal state transfer*.

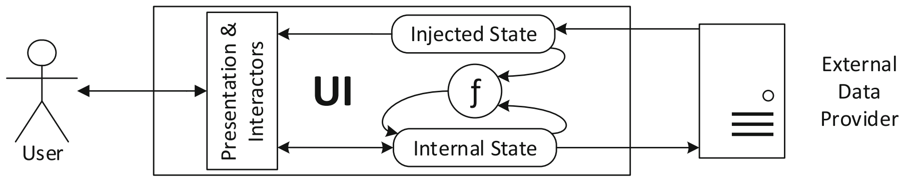


Fig. 1. Decomposition of the internal UI state

Taking the industrial demands for engineering and the UI characteristics into account, it is possible to put further restrictions to the MUI classifiers (cf. Sect. 2.2) without raising claim to completeness:

① **Initialization.** Frequently incoming or outgoing automatic migrations are conflicting with design guidelines (e. g. [12]) because of the likely negative impact on human operator's work. On the other hand, MUIs have the potential to enhance the support of the planning and conducting of maintenance work or to make the human communication more efficient. This might be true even if the migration is initiated manually. If applied, automatic migrations should rely on a set of well-tested rules. User-initiated migrations have the advantage of an easier implementation by omitting the rule evaluation plus a better situational awareness for the current tasks of the user and the respectively needed parts of the UI. This assumes that manual initialization also means that users select which elements to migrate. As an optimum between interfering with the target user's current work and exploiting the situational awareness of the source user, we propose a *migration relaying mechanism* as a hybrid method (cf. Sect. 2.2-①). This means that at first the source user has to notify the target user who may then actively pull the migration, i. e., no migration shall open itself automatically without the target user's request. The other way around, a pull that is initiated by the target user could be realized by asking the respective source user for his/her consent or, even more detailed, for the information he/she is willing to share within the limits given by authorization policies.

By adding a migration server as an *intermediate* location between the source and the target device, migrations are able to cope with interrupted connections. These intermediate locations would then act like chat servers that store migrations temporarily when transferring them from source to target.

In terms of directions of migrations, a pull mechanism (see Sect. 2.2) must prohibit the transfer of private data (such as control token, etc.) if not permitted by access rights of the user associated with the target device. In case of automatic triggering, the same restrictions apply to push mechanisms as well. In certain situations (process failures, etc.), and given the required access rights, a migration push may be required to overrule the above-explained migration relaying mechanism – which was intended not to open unrequested migrations on the target device – i. e., the migration system could be used to support the user in handling urgent situations.

② **Scope.** The access levels of the users also limit the set of migratable elements. If access to the plain/static UI is not restricted but access to process data providers in the background is restricted, total *and* partial migrations can be implemented without considering authorization issues. This will result in visualizations which are not updated completely in case a user has no complete read access. In cases where this poses risks to the process – because an important information is not available or only transferred once, but expected to be updated continuously, and thus, pretending normal operations – total migration is *forbidden*. On the other hand, partial migration, only limited to data that the receiver has access to, should not be safety-critical because the unavailability of required information is not hidden.

③ **Adaption Type.** If a migration requires adaption of the UI, either because of the changed device or incompatible access rights, both the adaption strategy and the result have to be predictable in order to ensure compliance with expectations given by standardization (see Sect. 3.1). It may even be necessary that the result of each migration has to be plannable in advance (during design phase).

In case of adaptions that are completely driven by algorithms at runtime, a set of adjustable formalized rules is needed. The applicability depends on the development costs necessary for the identification of relevant factors of influence and for the runtime gathering of each of the factors. If such rule evaluations are not performing well enough, such adaption mechanisms should not be considered.

Precomputed target UIs should always meet industrial requirements in terms of quality. If designed by domain experts, runtime adaptions are not necessary, but costs more or less depend on the number of precomputed UIs, as each of them has to be developed individually. If this approach is used, the migration engine requires means to identify corresponding elements across these UIs in order to navigate to the migrated elements on the target device. This is realized by creating links from a source element to its counterparts in other UIs.

④ **System Architecture.** The use of process data servers as central entities has already been indicated in Sect. 3.2. Typically, there are multiple tasks that these entities are delegated to, e. g., logging user interaction, alarm acknowledgement, user authorization, etc. The most important requirement when designing additional features like migration functionality is not to interfere with the server's real-time capabilities. This means that potentially expensive functionality (in terms of computation time) should be outsourced to separate instances. Concerning migration, we propose using a migration server and migration client structure in parallel to already existing UI clients and process data servers. Hereby, migration clients integrate with each UI in order to provide access to its state and means for users to control migrations (e. g. in case of user-initiated migrations, parameters like the target device have to be selected).

⑤ **Simultaneous Usability.** The ability to synchronize selected parts of the internal state of different UIs, for example, in a collaborative scenario, possibly results in the best performance regarding continuous usage. Because, once migration and synchronization have been initiated, users are able to swap devices without the need to *re*-initiate state transfers.

Multi-user scenarios may affect safety-critical behavior of a UI if, for example, the state of an input element is accessible by a second person without knowledge of the person who is in control of the process. Synchronizing certain UI elements continuously can be seen as a *hidden* control token transfer or token split onto multiple users which is forbidden in case they are not aware or not allowed to control the process. However, it is not unreasonable to synchronize certain UI aspects: For example, an operators view in a control room could be synchronized with a corresponding view on a field device – limited to non-safety-critical elements or to a read-only mode on the device without the actual control token – in order to achieve a four-eye-control or simply improved means to communicate. Non-safety-critical elements are, for example, navigators (synchronizing the view in focus) or in- and output elements for historical features like trends.

In a single-user and multi-device scenario, a synchronization of safety-critical UI elements is risky if the user assigned to a certain device is not in its physical operating range and thus not able to observe its inaccessibility by other persons. A mechanism for an automatic locking of the respective safety-critical function depending on the user's position could be applied in order to prevent harmful interference.

In consequence, a migration feature that enables simultaneous use requires an additional markup of the relevant internal state allowing or disallowing its synchronization, and the direction in which to synchronize, e. g., in a read-only mode on the source or target side.

Although, we discussed only 1-N and N-M scenarios, N-1 relations do not seem to be unrealistic. Such a scenario would require means to quickly and reliably identify and authorize the user triggering a safety-critical action out of all simultaneously acting users.

3.4 Summary

In this section, we discussed the implications on functional aspects of migratory UIs in the industrial domain. In order to enable UI migration, such UIs need to be enhanced with markup (1) concerning functional dependencies between UI elements with respect to the underlying process, (2) telling relevant and non-relevant parts of the UI state apart, and (3) allowing the configuration of the synchronization in multi-user/-device scenarios.

For automatic initializations and runtime adaptions, algorithms are required that evaluate formalized rules. This depends on the availability and reliable gathering of relevant factors of influence. Furthermore, it is important to be able to easily adjust such algorithms in order to ensure predictable behavior.

In case of manual triggering of migrations, user-selected migration parameters, and the use of precomputed UI adaptions, we propose data models containing the three types of markup that we have mentioned plus information about corresponding UI elements across the precomputed UIs (UI links). These data are created during design phase and are used for configuring migration engines at runtime. Hence, they have to be manually amendable.

4 Case Study

4.1 Existing Migration Engines

The analysis of available case studies and respective underlying designs reveals a limited applicability in the industrial domain. Tools for dynamic web migration [7] and TERESA-based applications [2] include a set of tools for runtime analysis of existing UIs in order to compensate missing explicit markup of migration-related information. However, it is not explained whether rules can be easily adjusted in order to amend the algorithm's results. The use of precomputed UIs as described in Sect. 3.3-③ seems not to be supported. MASP is another solution [4] with abilities to plan situations (different distribution scenarios regarding the available devices) in advance. The adaption mechanisms are not clarified, though [11]. The DireWolf framework [8] gives an idea of how to synchronize the state between different UIs but not with respect to the usual functionality of industrial process visualizations (see Sect. 3.2). Most of these approaches neither distinguish between relevant and non-relevant parts of the UI state, as introduced in Sect. 3.3, nor do they allow by design the detailed amending of migratory behavior. In case of using rule based algorithms at runtime, they have not been evaluated for the domain of industrial automation.

4.2 Concepts

The design of our case study ensures compliance to industrial UI standards by using individually engineered and thus precomputed UIs (see Sect. 3.3-③). For the configuration of our migration engine, we designed a data model as proposed in Sect. 3.4 – a *migration model* – covering additional information about the functional dependencies between UI elements, the description of its relevant state, and how to transfer it.

This approach is independent of the concrete UI adaption strategy or tools selected at the design phase which may rely on independently designed UIs for each device or even on an automatic deduction of a specialized UI from a *pre*-existing version (e. g. transformation from desktop to mobile UI).

Figure 2 shows the basic components of our migration engine mainly consisting of an MUI server that relays migrations and synchronizations via connections to available migration-clients. For that, it uses the migration model and an authorization mechanism. Each migration client integrates itself into the UI with a technology-dependent plug-in in order to get access to the state properties (client-based approach). The migration client provides users with controls that are necessary for triggering the migration and defining its respective parameters, such as the migration target.

4.3 Migration Model

Figure 3 shows the concept of our migration model. The migration-relevant part of the internal state is indicated by explicitly tagging migratable elements and

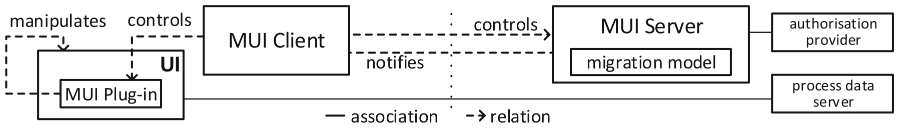


Fig. 2. Plug-in concept for enabling migration in common UI technologies

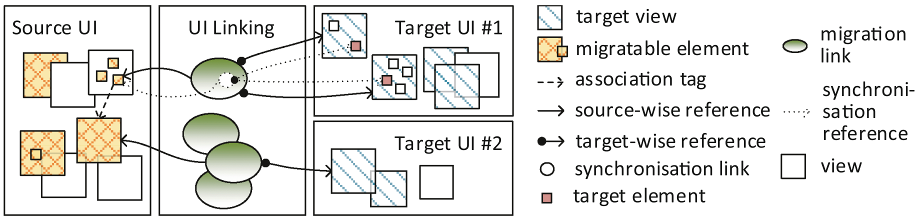


Fig. 3. Migration Model for UI linking and tagging

each of its relevant properties. The elements can be part of the presentational feature and the internal state as well. In addition, elements can be tagged as *associated* in order to reflect functional relations as discussed in Sect. 3.2. By defining such an association, the migration process can be influenced in three different ways: always migrate elements together (deny migration if not possible), migrate together if possible (otherwise allow separate migration of elements), or forbid simultaneous migration. If a UI is intended to act as a migration target, its respective presentations need to be tagged in order to determine distinct migration results. Analogously to the relevant state within the source UI, the target state properties need to be tagged.

The connection, and thus the description of the migration procedure itself, is realized by a set of migration links each telling the migration engine which presentation to display for each migrated element with respect to the concrete UI that is opened on the target device. As a subordinate feature of each migration link, synchronization links refer to state properties of the source UI and to each corresponding target UI property. As a property of a synchronization link, the mode of synchronization can be defined as once/continuous and target-wise/source-wise/bidirectional.

Concerning migration links, synchronization links, and element associations, we presume each property to be either enabled or disabled at runtime in order to be able to provide situational sensitivity for the migration behavior. This could be realized by integrating additional logic evaluating process properties (e. g. the alarm status), the user (as discussed in Sect. 3.3-⑤), the devices (e. g. battery status), or the environment[3]. Thus, there may exist multiple – and thus ambiguous – migration links for the current set of elements to migrate but, for example, pointing to different migration targets or including different synchronization links. Hence, an adaptation mechanism may also change the migration

[3] consistent with the context of use as described in [5].

model and thus the migration behavior, for example, by influencing the property of being enabled/disabled of different migration links.

4.4 Results

For the implementation of the migration engine, we decided to concentrate on user-initiated migrations only including the preceding selection of UI elements to migrate and of the migration targets. Possible migration targets are (1) devices that are assigned to the user initiating the migration and (2) other users. If a certain device is selected, the migration is pushed directly to this device. In case a user is selected, the migration will use the hybrid push/pull method as proposed in Sect. 3.3-①. The respective controls can be seen in the upper part of Fig. 4. If a user wants to select elements for migration, a UI overlay gets activated which shows migratable elements with half-transparent boxes in order not to overlap with any important information that might be displayed. The boxes can be clicked (switching to highlighted state – represented by the symbols A1-A3 in Fig. 4) in order to select the respective element. In the example, we selected a pump (cf. symbol A1 in Fig. 4) and a connected valve (A2) as well as the associated control elements (A3) for migration. This could be applicable in a service scenario where controls for a damaged unit (the pump) need to be transferred to a mobile device for in-field repairs as proposed in Sect. 3.2. The other components might be useful for testing after repairs have been completed. A migration can be initiated by hitting the "migrate" button (cf. symbol B in Fig. 4) with an optionally activated continuous synchronization feature (check box "keep synced"). On the target device (see Fig. 5 – left part), the user gets notified by the display of the incoming migration in a message box. After opening the message, the user is able to select an available UI for his target device and for his respective authorization level.

The message box could also be seen as a task-sensitive navigation hub, i. e., it provides special navigation shortcuts to panels that could all be required for a certain task. Normally, the user would have to navigate on arbitrary (not task-specific) paths to these panels. In the example denoted in Fig. 4, the source user could have assigned the task (e. g. to check the devices that belong to the migrated elements) to the target user.

In the example depicted in Fig. 6, a different UI has been opened compared to the migration source. By changing the layout of the UI, we simulated an adapted version for mobile devices.

5 Summary and Conclusions

In this contribution, we presented the fundamental basics of MUIs and discussed constraints and requirements to the UI development in the domain of industrial automation. We continued by discussing consequences for selected MUI properties. Finally, we introduced our case study including a migration model which is intended to work as an intermediate between MUI development on the one hand

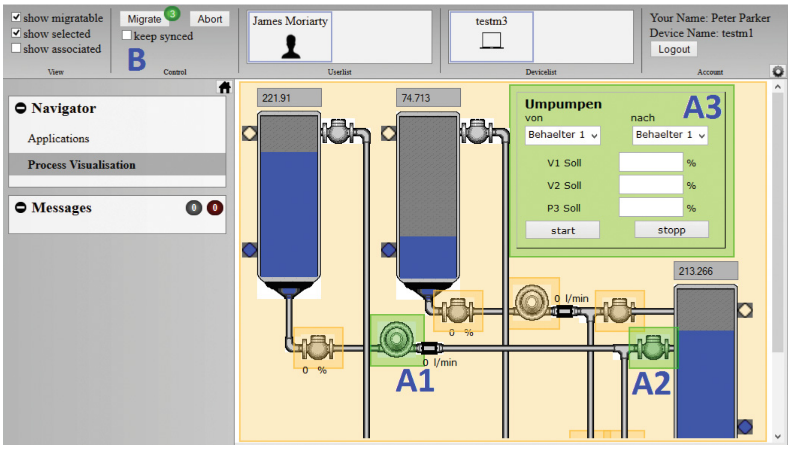
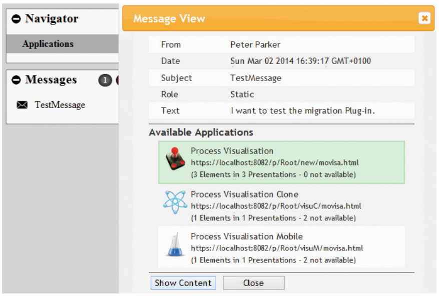


Fig. 4. Migration client with its controls (upper part) and an opened process visualization (right part)



From	Peter Parker
Date	Sun Mar 02 2014 16:39:17 GMT+0100
Subject	TestMessage
Role	Static
Text	I want to test the migration Plug-in.

Process Visualisation
https://localhost:8082/p/Root/new/movisa.html
(3 Elements in 3 Presentations - 0 not available)

Process Visualisation Clone
https://localhost:8082/p/Root/visuC/movisa.html
(1 Elements in 1 Presentations - 2 not available)

Process Visualisation Mobile
https://localhost:8082/p/Root/visuM/movisa.html
(1 Elements in 1 Presentations - 2 not available)

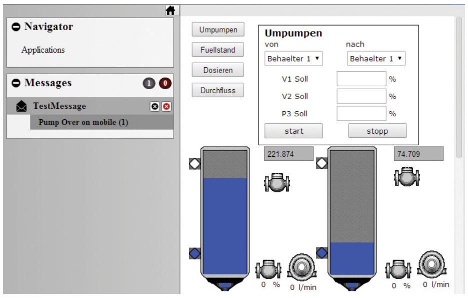


Fig. 5. Incoming migration notification on the target

Fig. 6. Opened target UI (differs from the source UI

and migration engines at runtime on the other hand. Despite the creation of the migration model being still relatively simple if the required domain expertise is available, it becomes difficult to manage in bigger projects and therefore should be supported by appropriate tools. If functional dependencies between UI elements are modeled well enough and thus the selection of elements to migrate becomes easy, we expect conformity of the migration behavior with user expectations. Better support for the user's current task is needed in order to be able to preselect UI elements for migration. Moreover, we intend to design a rule-based system in order to initiate migrations automatically.

By showing our first working prototype, we gave a sound indication for the potential of MUIs in industrial applications. By using a runtime architecture which integrates itself via plug-ins into the UIs, we enabled our migration engine for potential use in common industrial visualization applications. However, this kind of integration with existing visualization technologies requires proper public interfaces which the visualization system used in the case study does provide. However, in case another system shall be used, it yet has to be evaluated if these interfaces are present. Concerning authorization, our solution is based on restrictions to the whole UI instead of to UI parts or to single process data items. In the future, we also want to realize a fine-grained authorization and a token-based mechanism in order to dynamically control access when migrating.

This approach to migratory user interfaces in industrial process visualizations could be valuable in the future. Especially in the context of emerging paradigms which are propagating flexible production processes and logistics, such as *industry 4.0*, where user interfaces with migratory features can be an outstanding supplement to *plug&produce* scenarios, for example.

References

1. Balme, L., Demeure, A., Barralon, N., Calvary, G.: CAMELEON-RT: a software architecture reference model for distributed, migratable, and plastic user interfaces. In: Markopoulos, P., Eggen, B., Aarts, E., Crowley, J.L. (eds.) EUSAI 2004. LNCS, vol. 3295, pp. 291–302. Springer, Heidelberg (2004)
2. Bandelloni, R., Paternò, F.: Flexible interface migration. In: Proceedings of the 9th International Conference on Intelligent User Interfaces. IUI 2004, ACM (2004)
3. Berti, S., Paternó, F., Santoro, C.: A taxonomy for migratory user interfaces. In: Gilroy, S.W., Harrison, M.D. (eds.) DSV-IS 2005. LNCS, vol. 3941, pp. 149–160. Springer, Heidelberg (2006)
4. Blumendorf, M., Roscher, D., Albayrak, S.: Dynamic user interface distribution for flexible multimodal interaction. In: International Conference on Multimodal Interfaces and the Workshop on Machine Learning for Multimodal Interaction. ICMI-MLMI 2010, ACM (2010)
5. Chen, G., Kotz, D., et al.: A survey of context-aware mobile computing research. Technical report, Technical Report TR2000-381, Department of Computer Science, Dartmouth College (2000)
6. DIN EN 9241–110: Ergonomics of human-system interaction - Dialogue principles (2008)
7. Ghiani, G., Paternò, F., Santoro, C.: On-demand cross-device interface components migration. In: Proceedings of the 12th International Conference on Human Computer Interaction with Mobile Devices and Services. MobileHCI 2010, ACM (2010)
8. Kovachev, D., Renzel, D., Nicolaescu, P., Klamma, R.: DireWolf - distributing and migrating user interfaces for widget-based web applications. In: Daniel, F., Dolog, P., Li, Q. (eds.) ICWE 2013. LNCS, vol. 7977, pp. 99–113. Springer, Heidelberg (2013)
9. Lachenal, C., Coutaz, J.: A reference framework for multi-surface interaction. In: Proceedings of the HCI International (2003)
10. Paternò, F., Santoro, C.: A logical framework for multi-device user interfaces. In: Proceedings of the 4th ACM SIGCHI Symposium on Engineering Interactive Computing Systems. EICS 2012, ACM (2012)
11. Paternò, F., Santoro, C., Spano, L.D.: MARIA: a universal, declarative, multiple abstraction-level language for service-oriented applications in ubiquitous environments. ACM Trans. Comput.-Hum. Interact. **16**, 19:1–19:30 (2009)

12. VDI 3814–7: Building automation and control systems - Design of user interfaces (2012)
13. VDI, VDE 3699–3: Process control using display screens - mimics (2014)
14. VDI, VDE 3850–1: Development of usable user interfaces for technical plants - Concepts, principles and fundamental recommendations (2014)