Modeling Migration Projects
An ArchiMate and Paradigm case study in a university context

Nico de Groot (n.c.degroot@uvt.nl)
Supervisor: dr. Luuk Groenewegen (luuk@liacs.nl)

August 28, 2010

Leiden Institute of Advanced Computer Science (LIACS)
Leiden University, The Netherlands

Abstract

This thesis presents the use of the modeling framework ArchiMate and the coordination language Paradigm for real-life migration projects. An attempt is made to integrate the Paradigm language in the framework, to express migration processes inside the architecture. This makes it possible to specify future architectural changes and migration processes that implement these changes. This technique is useful to design on-the-fly migrations, this is illustrated using a real-life migration project, integrating two separate network infrastructures.

Acknowledgments

This work was done under supervision of dr. Luuk Groenewegen from LIACS, Leiden University. Many thanks to Luuk for the support and interesting talks we had about Paradigm, why it seems to be both simple and complex, and this project. The ArchiMate modeling tool BizzDesign Architect was provided by Harm Bakker, BizzDesign.

I would also like to thank my wife Karina for all her love, patience and care which really helped me to complete my study successfully. And my young kids, Willemijn and Nick who asked *very* often: Hoelang ben je nu klaar met studeren? / How long are you ready studying?

Contents

1 Introduction

This thesis presents and discusses the added value of using ArchiMate and Paradigm in combination in an practical, university context. The concepts from both modeling tools are used to describe and model infrastructural situations as well as a migration process. In general the value of using the ArchiMate language for modeling the *as is* and *to be* situations has been established. Additional snapshots can be made for the phases of a migration. And Paradigm can be used to model a dynamic migration. But how can Paradigm modeling be used in a practical migration situation? And the main question will be: Can we integrate the Paradigm/McPal modeling in the ArchiMate approach?

A typical migration process includes the evaluation of a limited number of migration paths. It is not easily possible - or even necessary - to determine all possible migration paths. It is clear that modeling can help to get a better view on the matter. How can we illustrate and model the migration processes with Paradigm? Are these models of value in an actual migration process? For which audiences (stakeholders) are the models usable?

An common migration scenario is the halting of the old situation and restarting in the new situation. The new situation has been prepared and tested in advance, without compromising the current way of working. The old situation is sometimes kept in a restorable state, to be able to switch back in case of unforeseen problems. In this top-to-bottom way of migrating, the necessary coordination is concentrated in a limited period. At the other end of the spectrum of migration strategies is an on-the-fly migration, coordinated in a more loose but still controlled way.

The on-the-fly migration is more complex and needs careful planning of the coordination. Are Paradigm and McPal useful tools in practice? What are the estimated costs and benefits? Can we capture the decision-points and the migration steps in a rule based system in such a way that the quality of the migration is improved?

2 Layout

The section 3 *ArchiMate and Paradigm* will introduce the concepts of the ArchiMate language and Paradigm. In section 4 *Analyzing the design of an actual migration process* we will record and analyze the migration paths that have been constructed and discussed in an informal(trial and error) way using ArchiMate models. In section 5 we reconstruct the migration using ArchiMate and Paradigm models not only for the migration but also for the migration design process. We develop a methodology for migrations and show the benefits of the Paradigm approach.

3 ArchiMate and Paradigm

Introduction of the concepts, the relation between ArchiMate, Paradigm and the use of the UML language and notation. The ArchiMate chapter is based on [1, 2]. ArchiMate is an Open Group approved standard (1.0) [3]

3.1 ArchiMate

3.1.1 The Language

The central idea behind the ArchiMate approach is to provide a well defined vocabulary to describe the design of enterprise architectures, to communicate it using different levels of detail and focus and to realize the design. An architecture is always a composition of parts and connections, and should provide views on the components and relations from different levels of detail and from the perspectives of different stakeholders. The quality of the architecture depends on managing this complexity and keeping all views consistent. ArchiMate uses visual elements from the well-known UML language, version 2.0, as notation for its concepts, but focuses on the meaning of each concept by specifying the formal semantics. Architectural migrations are specified by building separate *as is* and *to be* models, and possibly intermediate *snapshots*.

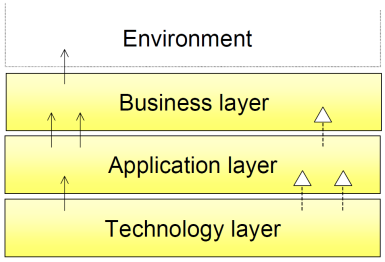


Figure 3.1: The three layers of ArchiMate

The Three Layers Figure 3.1 shows the layers of a complete architecture. The architecture is a model of a real enterprise or organization and describes the technical (IT) infrastructure, the applications in use and the business processes, all in the context of the environment of the enterprise. This environment include the suppliers, the customers and all other stakeholders.

The *business layer* provides products and services to the customers. They are realized by business processes that are performed by the *actors* working in the organization in certain *roles*. These processes are - in most cases, not always -dependent on software applications used in the enterprise. Put in ArchiMate terms, this layer is supported by the application layer which provides *application services*. And this layer rests on the infrastructural services provided by the technology layer. For an overview of all language elements see [2]. In this layer we model the network architecture using *nodes*, *devices*, *networks* and *communication paths* and *relations* like *composition, flow, realize, use* and *association*. In section 3 we show complete examples situated in the technological layer.

Modeling an organization's network infrastructure or the components of the applications in use, in ArchiMate is relatively straightforward, we use familiar names and relations. They are examples of modeling a static structure, but we ArchiMate can also model the individual and collective behaviour using concepts like *interaction* and *collaboration*. We have already stated that we can model different perspectives (views) ranging from internal to external and we can focus on individual actors or on more higher level entities. Figure 3.2 shows the three dimensions and expresses the relation between the elements. Services are realized by elements that perform behaviour like actors and devices. At the same time these element have a structural aspect. Services can depend on individual elements in an *interaction* or on a collective behaviour in a *collaboration*. Services are formally specified and accessible through *interfaces*.

At first sight the ArchiMate approach appears to be usable mainly for industrial and technical enterprises, but as our study will show it is very suited for more service-oriented organizations like universities. The central concept of *providing services* in this chapter already points in this direction.

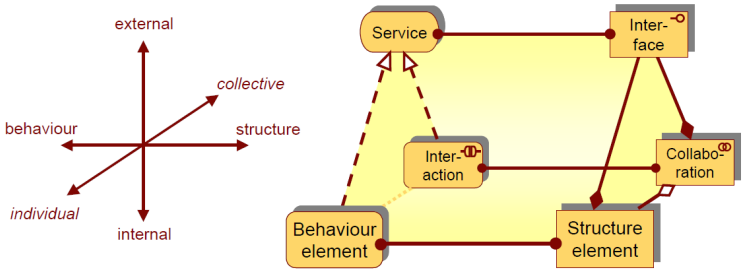


Figure 3.2: ArchiMate concepts

3.1.2 Modeling migration in ArchiMate

Usually for migration projects multiple models are made. At least an *as is* and a *to be* version and possibly intermediate *snapshots*. The models can be placed on a time line. When taken to the extreme a migration process can be modeled by creating a new snapshot for each group of related changes or even each individual addition or removal. The models can be very helpful to analyze the impact of each change.

Using ArchiMate tools like BizzDesign Architect help to build models and views by providing drawing tools that are aware of the language constraints and by storing the elements in a central repository. As elements are often reused in other diagrams, this make it easier to keep the diagrams

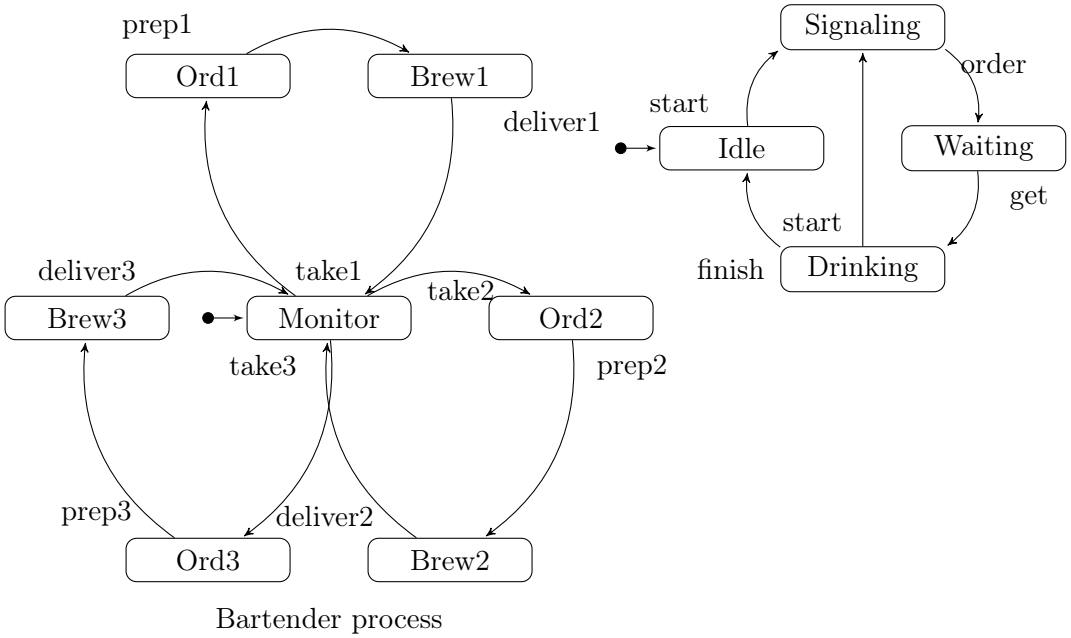


Figure 3.3: Detailed behaviour Customer$_i$ and Bartender component

consistent.

3.2 Paradigm

Paradigm is a coordination language to specify the behaviour of components and the coordination between components. The basic level of behaviour of a component is modeled using a state-transition diagram STD. It contains a set of states S, a set of actions or transition labels A and the set of transitions T such that $T \subseteq S \times A \times S$. A transition (s_{from}, a, s_{to}) can also be denoted as $s_{from} \underrightarrow{a} s_{to}$The current state $s_t \in S$ and s_0 is the starting state of the STD$_{t=0}$. Taking a detailed step or action or *firing* the transition $s_{from} \underrightarrow{a} s_{to}$is an atomic operation which can only be performed if the current state is $s_t = s_{from}$. Afterwards the state is $s_t = s_{to}$. The actions taken model the *detailed behaviour* of a component.

The Paradigm language uses a special view on this behaviour to specify the coordination. This view uses a partition of the states (and transitions) to model subprocesses. To give a minimal example: we have two types of components, three costumers and a bartender with his coffee brewer. The customers can be modeled using the states *Idle, Attention, Waiting* and *Drinking*. See figure 3.3The states of the bartender are *Monitoring, Ord1,2,3* and *Brewing1,2,3*. The *Ord$_i$* states corresponds with an order given by consumer i and the states *Brew$_i$* can be left only when the coffee is ready and delivered.

Our bartender - a manager type component in Paradigm terms - has a short attention span, and he can only handle one order at a time. From his standpoint each client has one of two - more global - states: *CanOrder* (customer can decide to ask for service) and *Waiting* (customer is waiting for his order). The states correspond with two subprocesses in figure 3.4 and each subprocess contains a subset of the detailed states and transitions. The set of subprocesses form a partition which we call OM, order management. The bartender doesn't need to know the actual detailed state the consumer is in, but some information about the behaviour is necessary. And the costumer - at least if she decides to want a drink - needs two interactions with the bartender otherwise the *workflow* halts. In subprocess CanOrder the customer cannot go further than the state Attention and she can not leave the trap until she has ordered, then she is again stuck in the state Waiting. To model this we can assign *traps* inside each subprocess. A trap is a non-empty set of states, all of its states belong to the detailed states and

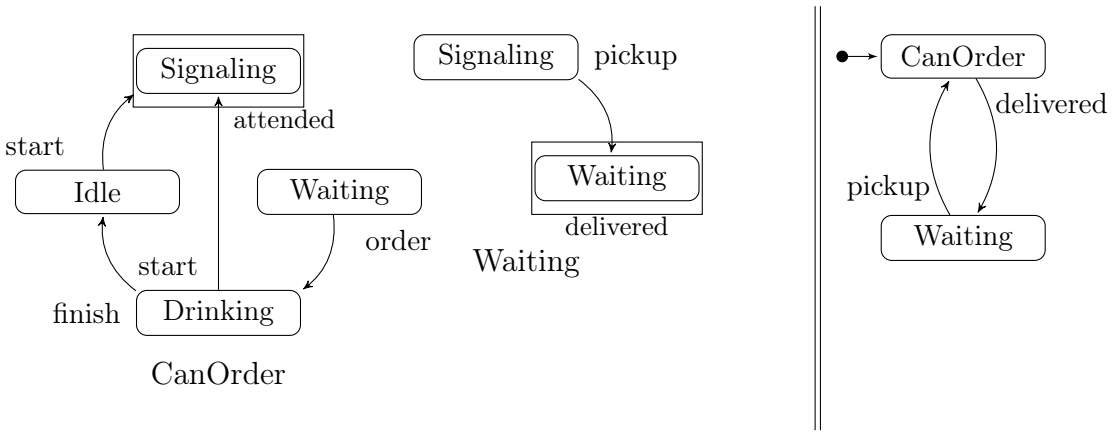


Figure 3.4: Subprocesses of the partition OM of the consumer process and the global process

the trap has a name. A trap is *connecting* when the states of the trap are part of another subprocess' state space. A connecting trap models a change of the global state, and thus can be seen as a global transition, with the name of the trap reused as the name of this transition. From the *detailed STD* we have now created a compatible *global STD* which models a *global behaviour.* These transitions are the building blocks to model the interaction and coordination between the components. The subprocesses and the traps show the constraints on the detailed behaviour and to specify the coordination details Paradigm uses *consistency rules.* These rules describe the detailed steps in textual form and specify the coordination. The detailed steps of the consumer can be formulated as follows:

- Consumer(i): Idle $\underrightarrow{start}$ Signaling
- Consumer(i): Signaling $\underrightarrow{order}$ Waiting
- Consumer(i): Waiting $\underrightarrow{pickup}$ Drinking
- Consumer(i): Drinking $\underrightarrow{start}$ Signaling
- Consumer(i): Drinking $\underrightarrow{end}$ Idle

The second form of consistency rules couples a detailed step of the manager-component with one or more global transitions. We give two examples, note that the rule set of the Bartender is not complete.

- Bartender: monitoring $\underrightarrow{prepare}$ Ord_i * Consumer_i(OM) CanOrder$\underrightarrow{attended}$ Waiting
- Bartender: Brew_i $\underrightarrow{deliver}$ monitoring * Consumer_i(OM) Waiting$\underrightarrow{deliver}$ CanOrder

At starting time the Bartender is in state Monitoring and all customer in the detailed state Idle and global state CanOrder. This coordination is minimal. One coupled action is the Bartender's starting to prepare a drink for costumer_iwhile at the same time the constraints of this customer changes, switching to the subprocess Waiting. The trap can now be left and to Waiting entered. The other costumers can't enter other subprocesses, but within the restrictions of their current subprocess they remain free to change their state. This is the case because the Bartender can eventually only go from state Brew_i back to monitoring releasing the waiting customer to the subprocess CanOrder.

Or in other words, we just specify that one of customers get serviced at a time. The detailed steps of the Bartender guaranty that during the brewing process no other costumer can change its global state to Waiting. We can't state that the selecting process is *fair.* Any other customer may have to wait very long. Refining this checking process is possible, implementing a round robin method of selecting is possible and we can even use automated verification of Paradigm models by translating them to process algebra to prove properties of a model.**(author?)** [4]. For a full introduction to the Paradigm approach see **(author?)** [5].

Note that the enforced (or committed to) constraints of the subprocess and traps do not *directly* change the state of the managed process. The autonomy of the managed process is respected (not

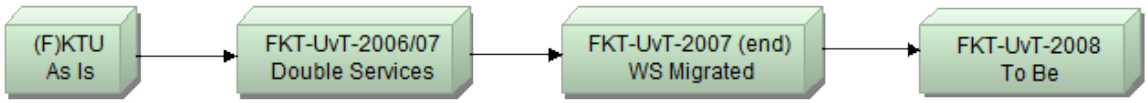


Figure 4.1: The global infrastructural phases

unnecessarily restricted) while at the same time old policies are removed and new policies enforced. This modeling approach elegantly supports the separation of responsibilities at the different levels of managing multiple concurrent work processes.

More complex examples of Paradigm model are used in the next chapters, including models that have the ability to change themselves .

3.3 Integration of Paradigm in ArchiMate

The Paradigm approach gives us interesting possibilities to specify collaboration and coordination protocols and, as we will see later models in migration. The ArchiMate language can express collaboration and interaction on a global level, but it is not designed as a coordination language. We will investigate how the two approaches can be used together.

4 Analyzing the design of an actual migration plan

4.1 The case

Our case study is the migration and integration of the IT infrastructure, applications and work processes of a formerly independent University (Catholic Theological University, the KTU, located in Utrecht on the campus of the Utrecht University in the Netherlands) to and in the infrastructure, applications and work processes of the University of Tilburg (UvT). The new name of the institution is Faculty of Catholic Theology (FKT). The merge has taken already place administratively, but the workstation migration is the first major step in the IT migration. The users and workstations will remain at the location Utrecht, but must be able to communicate with the Tilburg infrastructure in the same way as the users and the workstations physically on the campus in Tilburg. The use of local servers is to be minimized.

The technical solution chosen is to use a *Lichtpad connection*, a glass-fiber connection provided by SURFNET which enables a fast and shielded connection to the Tilburg network. On the Utrecht location this connection is distributed to the servers and workstations by using the existing virtual network infrastructure(VLAN) of the University of Utrecht.

The phases of the integration are shown in figure 4.1.

The first phase is the original situation: all services (workstation and server) are local[1]. In the last phase, the infrastructure has full UvT services and minimized local services. Phase two and three - the arbitrary intermediate phases- will be designed and explained later. It is clear that we will be changing the infrastructure and the workstation configuration to enable access to the Tilburg file, print and application services. The exact steps will have to be designed and evaluated in advance. First we will describe *what* has to be done and later *how* we can do it, designing comparing different migration paths.

4.2 The workstation migration, technical background

There are a few complicating factors. We will not elaborate on all of the solutions, most fall outside the scope of this document.

[1]The salary administration is outsourced. This service shall be taken over by the Tilburg University. This migration will not be considered here. Another migration project not modeled here is the partial transfer of the student administration to the Tilburg location.

Network connection

The physical network connection will not change. By changing the original VLAN of a node (workstation) in the Utrecht network to the Lichtpad VLAN the node can be switched to the Tilburg network.

Log-in and, file and print-services

The KTU uses a Microsoft Active Directory infrastructure (Windows Server 2003) and the Tilburg University uses a Novell and Unix infrastructure. So not only is the location of the authentication-service different, but the client-services need to be fully replaced.

Difference in roaming facilities

The KTU uses *roaming profiles*, the user environment, the desktop and settings, is independent of the choice of workstation, the Tilburg workstations do not support roaming. Only the email client configuration gets adapted at log-in.

Access to local library facilities

The KTU is partner of the Utrecht University library and its researchers use the electronic facilities which use identification by IP-range. As changing the VLAN also changes the IP-range, this will block the access.

Limited access to the Tilburg VLAN structure

Because of local infrastructural limitations only *one* Tilburg VLAN can be mapped to the Utrecht VLAN. Access to other VLANs (IP-telephony, server VLAN) is not possible.

Summary of the workstation changes

On each workstation the VLAN and the network client and application software on each workstation must be changed from the FKT to the UvT configuration. Local data on the workstation has to be retained.

4.3 The requirements for the migration

Performing the migration should only have a minimal impact on the FKT employees. The management has formulated the following requirements.

- Minimal or no disruption of the work processes.
- All applications should be available in the new situation.
- No loss of local and group data. 'Everything' has to be transferred.
- No versioning problems during the transition.
- Adequate local support for the users during the migration.

It will become clear whether all the requirements can be fully honored.

4.4 The as-is and the to-be situation

At the beginning of the migration design process, two configurations are clear, the as-is situation (figure 4.2) and the situation to-be (figure 4.3). The snapshots have been made using the ArchiMate modeling language using the tool BizzDesign Architect.

In the as-is situation there are three locations, shown by the grouping image, the *FKT(U) location at Utrecht, the UvT location at Tilburg* and the location *elsewhere* to indicate access to the different services from locations like the home address of the FKT(U) employees, or other Internet connected

locations. At the top level we find the services available for the applications and the work processes. At the Utrecht location this includes web services, authorization (the Active Directory), database service, CMS management functions, mail services and Internet services. The services are provided by three servers, running the processes External (DMZ) web server, Name server (DNS) on the first server *Dienaar01*, CMS web services (Zope), Database-, File- and authorization server on *Dienaar04* and the mail- and anti virus server on *Dienaar05.* Only one of the workstations is depicted. The router connects the local network to the UU network. There are two servers on this network that provide services to the FKT(U) namely the Blackboard server, providing the Electronic Learning environment and the server at the Utrecht Library providing applications and digital sources. This network (and indirectly the FKT(U) network) is connected to the national SURFNET Internet backbone.

At the Tilburg location some of the services like the digital sources of the Tilburg Library, management services and database services are already available without any infrastructural change. They can be accessed using a VPN connection, depicted here with 'UvT VPN'. The nodes at the Library UvT, the central UvT servers and the nodes of the Administrative center run servers like the Repository server, web application server, the UvT web server and the database for scientific publications Metis. The nodes at the Administrative center (DEA/DSZ) provide database servers for the student administration (SIS), the employee database (SAP/HRM) and the relations management system. These three (groups of) servers connect to the Tilburg network. This network is - just like the Utrecht University network - connected to the SURFNET backbone.

The to-be situation is the situation where the Tilburg file-, print- and application services are available and local services are minimized.

In the to-be situation we see that most of the services are transferred to the Tilburg location. The applications and the Blackboard, database and the file, print and authorization services now originate there. The essential change is possible because the *Lichtpad* connection provides direct access to the UvT infrastructure using a dedicated Internet tunnel connection. The access to the Internet is also provided using the Lichtpad, ultimately using the UvT router connection to SURFNET. The router connection to the Utrecht University network is now obsolete. New reachable servers at the Tilburg location include the data cluster Lyra and the UvT Blackboard server. The VPN connection is no longer needed at the Utrecht location, only at external (home) workstations. A few services remain located at the Utrecht location.

Comparing the two situations from the user perspective, we identified the basic migration steps.

1. Change the VLAN
2. Duplicate the user and group accounts in the Novell administration
3. Duplicate the user network data in the Tilburg file system
4. Duplicate the shared data in the Tilburg file system
5. Back up the data and settings on the local workstation
6. Install the new software configuration on the workstation (imaging)
7. Check and adjust the new configuration (user log in)

After testing and discussions a few constraints have been established.

- The availability and speed of the *Lichtpad* connection is essential.
- Changing the VLAN has to be performed by an external party, and scheduling this change for each connection separately is not feasible.
- Copying (and synchronizing) the user and shared data can be realized using an new service on a new double-connected server. Synchronizing at an earlier stage will reduce the turn around time of the individual workstation migration.

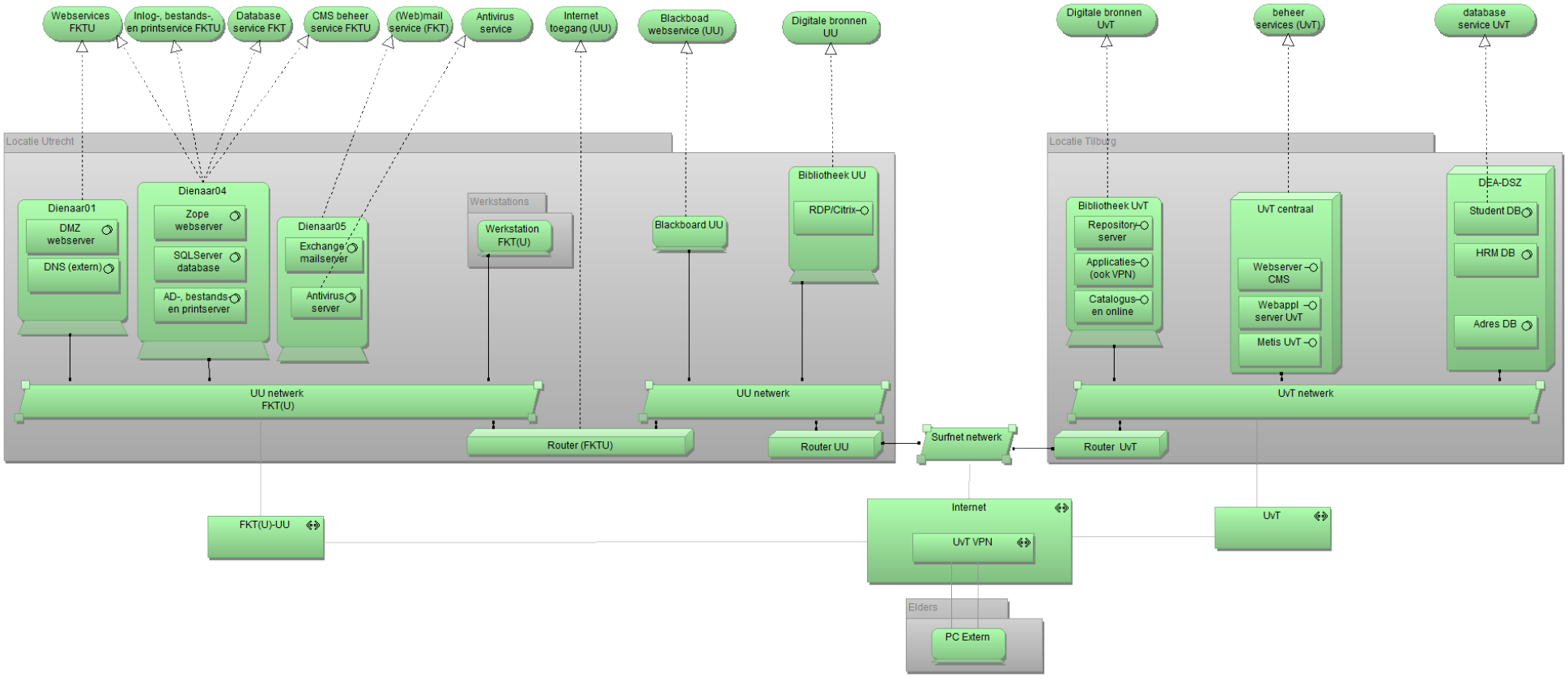


Figure 4.2: As-Is, local services

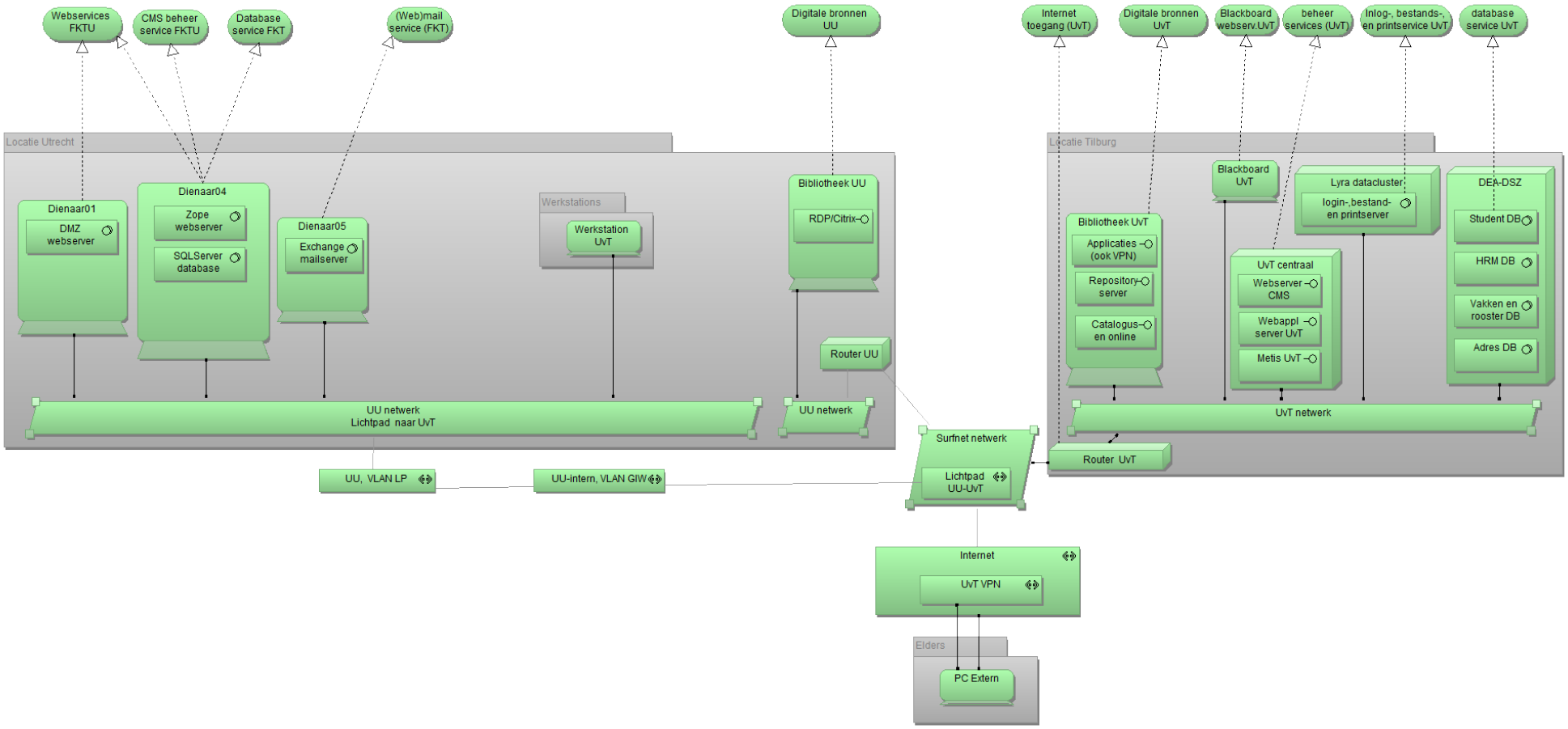


Figure 4.3: To-Be, UvT workstation services, minimized local server services

- Changing the software (backup of local data, install new operation system, software and updating) can take up to three hours[2] for each workstation, there are 60 workstations.
- Configuring the workstation after the migration needs some manual intervention by IT staff members.
- Availability of IT staff is essential at the moment the user logs on in the new environment, for support and instruction.

A number of approaches have been discussed:

1. Migration by halting, migrating and restarting all stations at once.
2. Phased migration, by department of by floor.
3. Migrate one workstation at a time.

When we analyze the discussion we find that we are in fact identifying coordination, constraints and dependencies. At the same time we are trying to design possible migration paths. The as-is situation is clear: All workers are using the local infrastructural services documented in figure 4.2. The employees use applications and services provided by the servers D01, D04 and D05 connected to the UU network. Some applications in the UvT infrastructure can be reached using a VPN connection. In the to-be situation, see 4.3, all workstation and server services[3] are provided by the UvT network.

The search for a good migration design has a backtracking aspect. When not all requirements are fulfilled, we go back and try to adapt the earlier steps or move steps between the workstation and the infrastructural level. We further evaluate this process by looking at it from different perspectives, to check the consistency and to minimize the overhead for the worker and for the IT staff.

The first migration strategy we discussed is a *brute force* migration path. We prepare the migration by creating all accounts in the new environment, logging off all users to prevent versioning problems, then copy the local and group data. Then, for each workstation, we backup the local hard disk, change the software, switch the VLAN (or schedule it) and reboot. The advantages of this path are clear, no individual coordination is needed, no versioning problems and the time needed to perform the migration can be scheduled outside business hours. But in our particular situation, the support is problematic: the scientific staff, for a major part, works at home. Getting all of them to be present the first day after the migration is not feasible. Expanding the support staff during the first days after the migration is an option, but does not completely solve the problem. Availability of support at the moment the staff member does appear is difficult to realize because of the minimal IT-staff after the migration. Another bottleneck is that a large number of the individual problems are expected to be discovered in the first hours after the migration. This will produce delays for the other workers with problems. And there is a risk involved. Backing up and transforming the workstations may take more time and handling than expected. Turning back halfway through the process is not a feasible option.

The second option, migration in groups, is only a variant, but this approach is already better because it limits the risks. But versioning problems regarding the shared data prevent this, the different groups will work side by side but cannot have write access to the same shared data. Coordination of the external VLAN switch and the workstation transformation is still problematic.

The third option has the advantage that the migration work is divided in manageable pieces. But this requires individual VLAN switches, which is even more problematic. So we ask ourselves if we can remedy the consequences for the current workstations if we perform this switch earlier. We were able to realize this by keeping the old services also available in the new situation, in effect adding a new infrastructural phase. And we now can also delay the migration of the shared files services, to prevent the version problems. In the view 4.4 a new local network is constructed, using a new local VLAN, connected to the already available Lichtpad connection (see also the to-be situation). This

[2]The Utrecht technical infrastructure, not the *Lichtpad*, is responsible for the slow performance, the bulk of the physical network connections are half duplex 10 Mb/s. In combination with the UvT installation and update process, which consists of transferring many small files this results in a worst case situation. The (few) full duplex 100 Mb connections perform much better, and reduce the installation part of the process with a factor 10 or more.

[3]The mail service is integrated in the UvT mail infrastructure, and provided by a FKT server

network is transparently coupled with the UvT network. Locally the FKT(U) server nodes are also connected to this new network using additional network cards. An additional *Image server* is added, see below, and an extra node *Migration machine* to the infrastructure to enable migration services like the synchronization of data. In the group *Workstations* we can see the three types of workstations available. The original *workstation FKT(U)*, connected to the original network, the identical *workstation FKT(U)/UvT* but now connected to the new *Lichtpad* network and the workstation of the future with the new set of software, the *workstation UvT*. Remark: the use of the services by the workstations is not drawn.

Creating this infrastructural phase gets the VLAN switch problem out of the way and opens the possibility to perform the workstation migration on the fly. To speed up the migration from the worker perspective, we decided to prepare a number of UvT workstations and use them to physically swap the workstation. In a temporary workshop the backup will be made and the machine will be transformed into a new UvT workstation. The procedure for the worker is the following.

1. The worker signals that s/he is ready to migrate
2. As soon as possible the worker receives a UvT workstation, the old one is taken in. (the workstation swap).
3. An IT staff member will perform the adjustments and give instructions.
4. The worker can continue the work.

In the background, the old workstations are handled:

1. An backup-image is made to be able to recover lost local files at a later date.
2. The workstation is transformed into a UvT workstation.

The 'new' UvT workstations are now ready to be used in the workstation swap for the next worker.

When all workstations are migrated we can remove the local workstation services and enter the next phase, the view 4.5. There is still a server connection to the UU network, to provide the connection from the Internet to the DMZ web server and the external DNS.

This step of going to the last phase (the *to be* situation) is to change and remove the last services and remove the now obsolete network connection.

This approach was successfully used in January 2008. Our investigation of ArchiMate and Paradigm will use this migration as a focus point. With hindsight we can analyze this scenario and maybe develop different scenarios using these (and some other more general) techniques.

4.5 First evaluation of ArchiMate, lessons learned

The infrastructural ArchiMate views proved to be very useful to document, discuss and design actual and future technical situations. Simplified snapshots can be used to illustrate infrastructural migration paths. These migrations are relatively simple and do not really need coordination modeling. We just add new services and network paths without interrupting the old ones, switch over the workers and in the end remove the old services. But when we consider our requirements, it proved to be better to look at the migration process from the viewpoint of the worker. When we look at the main concepts of the ArchiMate language see figure 4.6 from the ArchiMate book ([1]), we conclude that until now we have modeled the active structure in the technology layer. This is understandable, the applications, the objects and business processes do not essentially change.

But there is an exception: the migration process is a new business process. In our case, the design of this process was ad hoc and not very structured. The migration process was not modeled in ArchiMate, we used instruction sheets and checklists and adjusted them along the way.

Although the actual, successful migration was not modeled, it certainly was inspired by the on-the-fly migration examples in the Paradigm articles. This poses the question whether the design of the migration processes and the migration itself would have been improved if the ArchiMate and Paradigm approach was used fully from the start.

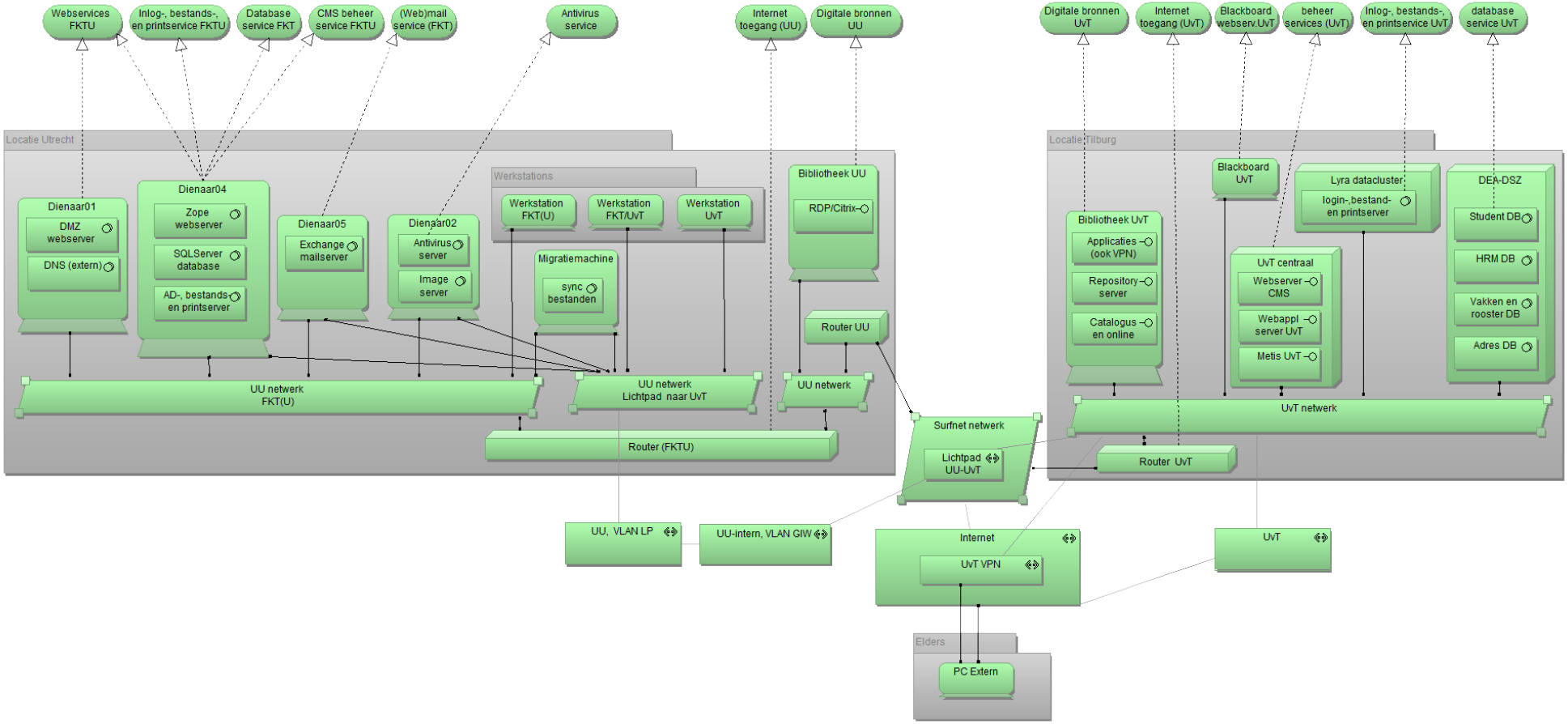


Figure 4.4: UvT and local workstation services

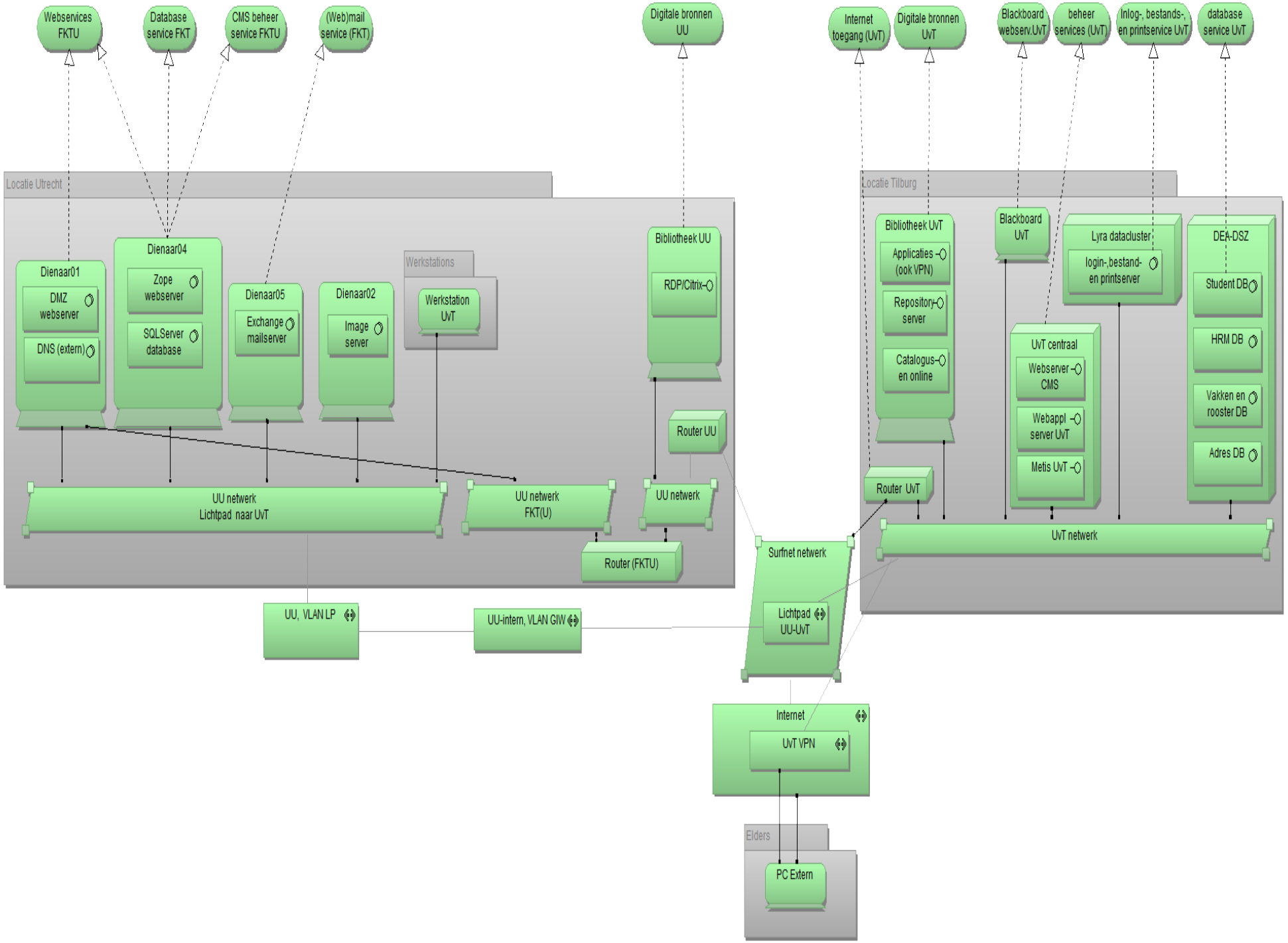


Figure 4.5: Only UvT workstation services

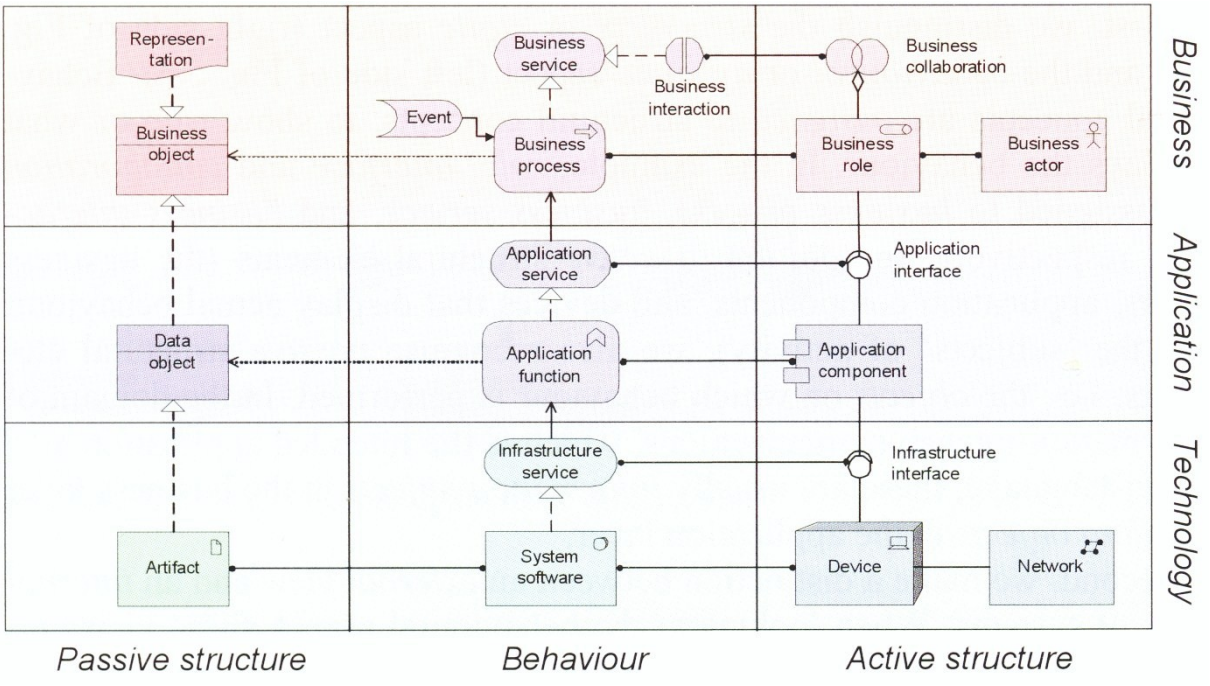


Figure 4.6: he three layers vs. behaviour and structure in the ArchiMate Language

5 Re-examining the migration using an ArchiMate and Paradigm methodology

5.1 Modeling migration processes

By modeling the migration in Paradigm terms we hope to improve the quality of the migration - as it took place - by separating concerns and explicitly identifying the coordination. We will compare this to the actual migration and analyze the differences. Secondly we want to integrate this model into ArchiMate.

Rephrasing the migration in Paradigm/McPal terms is the first step. We have identified that there are different *roles* in the migration process. The role *Worker* and the role *IT staff* are clear. But we should not forget the planning and coordination involved, the role *IT manager*. Each role has a different perspective on the migration steps. The coordination between the manager and the staff is invisible for the worker. And adding new services is a migration step at the manager level, invisible for the worker. The Paradigm model will show the detailed processes as well as the global picture, abstracting the detailed processes, showing the connections and thus integrating the different perspectives.

5.1.1 The Standard support procedure

But to start simple and to further introduce the Paradigm concepts, we first model a simpler case: the well-known IT support call. The actual migration shows that this is a good starting point. We first model it in Paradigm using the techniques and example of [6] and later transform it for the migration procedure. We will see that we can use to model this transformation. After that we remodel them in ArchiMate while trying to preserve the benefits of Paradigm model.

We first simplify the support procedure: In the normal situation a number of employees work individually on their workstation and no coordination is necessary. Each employee can signal 'I need support' which is immediately recorded in a queue managed by the IT support staff. This call is picked up from the queue sooner or later by one of the members of the IT support staff. In the meantime the worker can continue with another task. The work that needs to be done by the IT staff, the actual support (from a distance or locally on the work floor) can be seen as a critical section. After the support the worker process continues and the IT staff member is ready to take on the next support assignment.

This procedure is similar to the worker-scheduler process in the Paradigm articles (see e.g. [7]) but

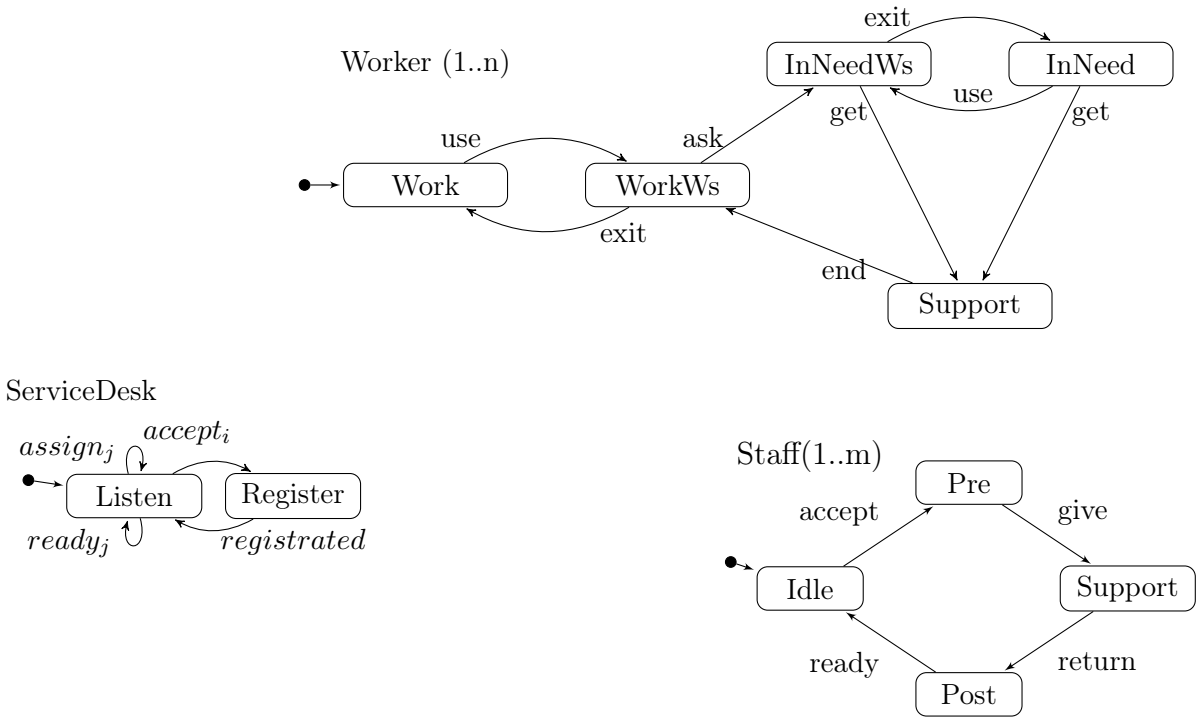


Figure 5.1: The worker, scheduler and support processes.

in general there will be more than one IT staff member. We first model the worker process, then the support coordination process and the support processes. After that we create the consistency rules. The first observation we make is that in Paradigm we do not detail worker or scheduler decisions within a process[4]. Whether this is a problem we will see later, in any case it helps to prevent making the model too detailed. As in any modeling attempt we have to decide exactly *what* we want to show. We focus on the coordination, the signal 'I need support', the initial response and the actual support.

Figure 5.1 presents each worker process, the scheduler and the support process as *state machines* using states and transitions. A state represents a situation, and a transition an action[5]. The workers have their own independent tasks, using a workstation (*WorkWs*) or not (*Work*). On a certain moment, while working with the workstation the worker may have a support question or may experience an IT problem he or she cannot resolve. In that case a *support call* is issued (and recorded) and the worker enters the state *InNeedWs*. The worker can continue on other non-related tasks or decide to work without the workstation in *InNeed*. In both states the worker is waiting for the support to be given. When the call is activated and actual assistance is delivered the state *Support* is entered, to be left again when the assistance has ended. The assistance can be seen as a critical section, any IT supporter gives support to one worker at a time.

The worker process can be divided into subprocesses. After the transition *ask* the worker is restricted to the states InNeedWs and InNeed. He can only leave these states after acknowledgment by the ServiceDesk. We model this constraint by using the Paradigm concept of a *trap*. A trap marks a set of states and transitions, that cannot be exited unless a certain external condition is valid. The name of the trap reflects the condition the worker is in. In this case the trap is called *requested*. This marks that a support request is made. A trap is visualized in the diagrams with a polygon around the state

[4]For example by using *guards*.

[5]As in any model this is an abstraction: a state can represent a situation from a observer perspective. Inside this state, when we further analyze, there are situations and actions, for example if we we take a worker perspective.

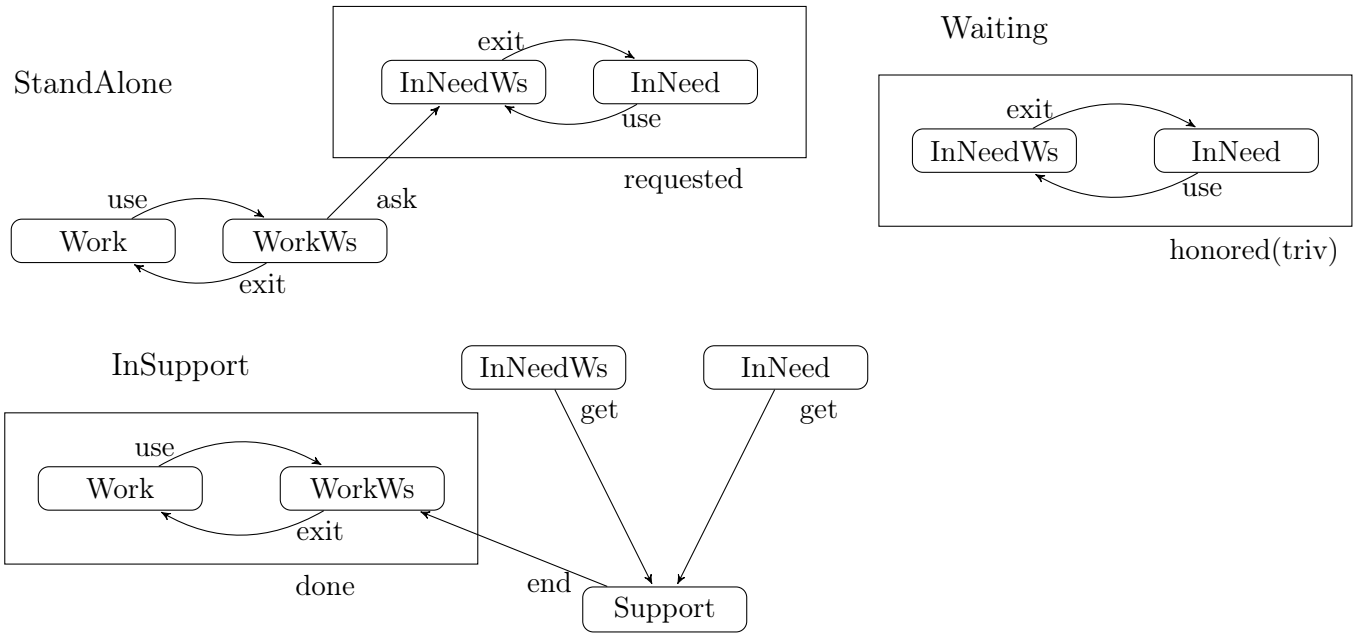


Figure 5.2: The subprocesses of each worker

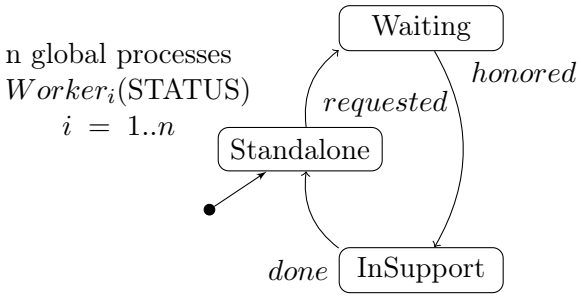


Figure 5.3: Global process of each worker, level STATUS.

(or states) it consists of. A subprocess is given a descriptive name, here *StandAlone* and ends in a trap. The state(s) inside a *connection trap* are also part of the next subprocess. A connecting trap connects the subprocess to another subprocess. This trap will be left when the IT service desk acknowledges the request, which moves the worker to the next subprocess.

We reflect in the next subprocess that the request has been registered and that the worker is waiting[6] for the request to be honored. We have created a separate subprocess *Waiting* to model this. The trap *honored* encloses the whole subprocess. And this trap can be left if Scheduler assigns the support call to a Staff member and the Staff member starts to *give* the actual support. The *InSupport* subprocess shows the actual support given, leading to the preferred situation in which the worker is again without request, working independently. So there are two coordination constraints that connect the worker process to the other processes. The first one links the worker to the service desk and the second one links the worker to the IT staff member. We show the subprocesses *StandAlone, Waiting* and *InSupport*, in figure 5.2.

From these subprocesses and traps we build the following global states and transitions. Each worker process has three global states, see figure 5.3. We see the traps of the subprocesses reappearing in the transitions. The global states can be seen as a high-level view on the worker process showing only the state-changes relevant for the external world. This provides a separation between the detailed level and

[6]The possibility that the worker finds the solution herself and cancels the call is not modeled here.

the external level, where the 'interfacing' with the other processes takes place. The detailed process can change in a number of ways, for example when a new detailed state is added between *Support* and *WorkWs*, or inside a trap a state is added, but the global states and transitions can stay the same. We will discover other advantages later.

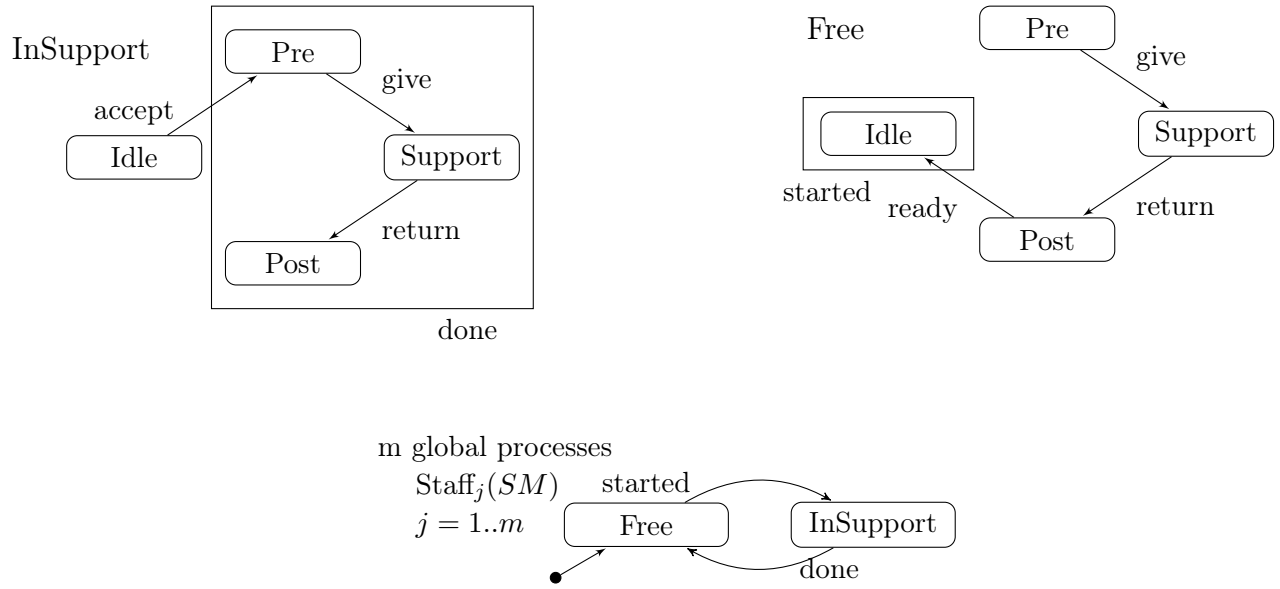


Figure 5.4: The Staff subprocesses and global processes

The role of the Service Desk is to register the request, assign a request to a staff member, monitor the support process and update the status of the requests and the support assignments. The staff member, as is illustrated in figure 5.4, after preparing in *Pre*, delivers the support to the worker, and reports back to the service desk. The support had to be modeled using delegation because there can be multiple requests for support and assignments to multiple staff members who deliver the support. We need a data structure to record the requests and assignments. A state diagram is not suited[7] to model a list or a queue, but this can be remedied by using local variables. A coordination constraint connects the staff member to the worker. When the actual support starts, signaled by the staff member going from *Pre* to *Support*, the global state of the worker changes from *Waiting* to *InSupport*. And the second constraint connects the staff member to the worker at the end of the support, but the service desk is also involved, recording the changed states of both the staff member and the worker in its local data.

Until now we have only used diagrams to illustrate our processes. Paradigm has a special textual syntax to describe the processes defining the states and the transitions, the subprocesses and traps, the changes of local variables and the coordination involved. The description consists of *consistence rules*. See table 1

The first eight rules define the states and transitions of the worker. The rules **s1-s4** describe the detailed staff process. The rule **s3** is special, it couples - using the '*' - the staff transition *give* (support), going from Prepare to Support, to the global worker transition *supported* going from Waiting to InSupport. The left side of the '*' is a detailed step in a process of one of the actors in a managerial role and the right side is a detailed or global step in another process. This rule describes the start of

[7]We could have modeled the schedulers local awareness of the states of the worker and the staff by creating extra states. There can be multiple requests and multiple active support-instances. It is possible to model a fixed size list of requests showing the status of each request, by creating a state for each combination of request states. But this generates a state explosion. For two requests we already need nine states. And in our model the length of the list is flexible and can be as large as the number of workers. In Paradigm - the textual notation - we can use local variables for the lists. If we really want to *show* the data structures in our visual model, we can use the UML concepts *guard* and *method* to to refer to the lists using methods like *addtoRequestList* and *removefromRequestList*.

w1	$Worker_i$:	Work	$\xrightarrow{use}$	WorkWs	
w2	$Worker_i$:	WorkWs	$\xrightarrow{exit}$	Work	
w3	$Worker_i$:	WorkWs	$\xrightarrow{ask}$	InNeedWs	
w4	$Worker_i$:	InNeedWs	$exit$	InNeed	
w5	$Worker_i$:	InNeed	$\xrightarrow{use}$	InNeedWs	
w6	$Worker_i$:	InNeedWs	$\xrightarrow{get}$	Support	
w7	$Worker_i$:	InNeed	$\xrightarrow{get}$	Support	
w8	$Worker_i$:	Support	$\xrightarrow{end}$	WorkWs	
s1	$Staff_j$:	Idle	$\xrightarrow{accept}$	Pre	
s2	$Staff_j$:	Post	$\xrightarrow{ready}$	Idle	
s3	$Staff_j$:	Pre	$\xrightarrow{give}$	Support	*
	$Worker_i$(Status):	Waiting	$\xrightarrow{honored}$	InSupport	
s4	$Staff_j$:	Support	$\xrightarrow{return}$	Post	*
	$Worker_i$(Status):	InSupport	$\xrightarrow{done}$	Standalone	
sd1	*ServiceDesk:*	Listen	$\xrightarrow{accept_i}$	Register	*
	$Worker_i$(Status):	StandAlone	$\xrightarrow{requested}$	Waiting,	
	ServiceDesk:[R:=R∪{i}]				
sd2	*ServiceDesk:*	Listen	$\xrightarrow{assign_j}$	Listen	*
	$Staff_j$:	Free	$\xrightarrow{started}$	InSupport,	
	ServiceDesk:[S:=S∪{i,j}, i=head(R) and R:=R\{i}], $Staff_j$				
sd3	*ServiceDesk:*	Listen	$\xrightarrow{ready_j}$	Listen	*
	$Staff_j$(Status)	Support	$\xrightarrow{done}$	Free	
	$Worker_i$(Status):	InSupport	$\xrightarrow{done}$	StandAlone,	
	ServiceDesk[S:=S\{i,j}], $Staff_j$				
sd4	*ServiceDesk:*	Register	$\xrightarrow{registrated}$	Listen	
st	($Worker_i$, $Worker_i$(Status)):(Work,StandAlone),($Staff_j$,$Staff_j$(Status)):(Idle,Free),*ServiceDesk*:Listen				

Table 1: Support consistency rules $Crs_{support}$

the actual support. In **s4** we can read that the support at the location of the worker is ended, the staff worker can now perform concluding actions in Post. Rule **sd1** uses a more complex rule. In a consistency rule it is also possible to show changes in local variables in a *change clause* with a notation using square brackets[8]. Here the request for support is received by the service desk, who registers the request in a local variable R. The service desk is on the left side and the worker action on the right side.[9] After registering the service desk returns to Listen (**sd4**). From this state the next coordinated step can be taken in **sd2**, assigning one of the requests to one of the staff members. The service desk records the assignment of the request of Worker_i to Staff_j in a local list S and the staff member records the identity of the requester in a simple variable W. Now the request is delegated to the staff member and can be removed from R. We have seen above the two rules for for the actual support. When the staff member is ready finishing the support in Post he reports back to the desk. Rule **sd3** reflects this. As we can see here, on the right side of a consistency rule it is also possible to give multiple steps, each in a separate process. We find here the staff member becoming available for a new assignment in the global state Free and a global worker transition going to the starting state Standalone. The bookkeeping is done in the local variables. Rule **sd4** describes the detailed step of the service desk returning to Listen after registering the call, with no coordination involved. In **st** the starting states

[8]Inside the brackets we use the mathematical set operators like intersection ∩, unification ∪, division \ and subset ⊂

[9]We could also switch the positions or put both transitions on the right side, and leave the left side blank, because in this case there is no real manager-worker relation. The chosen position hints that accepting multiple requests is a manager-like role.

of the models at the different levels is noted.

We see that the consistency rules enable us to formally specify different kinds of simple and more complex coordination, like processing requests, and delegation, involving multiple actors c.q. roles.

5.2 Modeling the design of the migration

When designing the actual migration process we created different ad hoc scenario's by identifying migration steps, and changing the order of the steps. Looking back we can model this process as follows in a (Paradigm) state model in figure 5.5. Because the design process is also a business process we also model it and add it to our ArchiMate model.

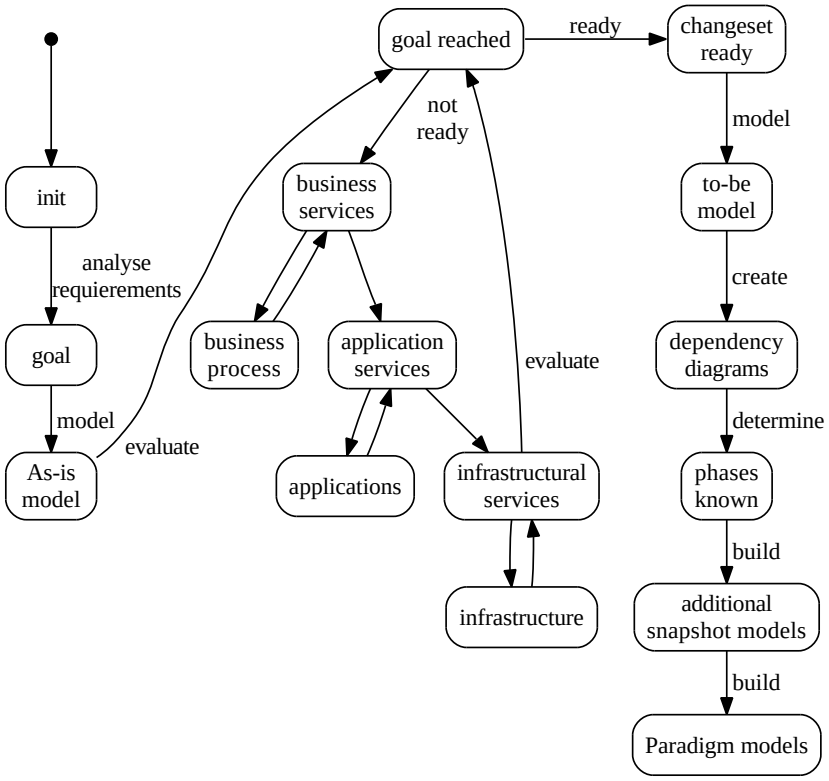


Figure 5.5: The migration design process (Paradigm).

From the requirements given we formulate a goal for the new to-be situation. In the cycle in the middle of the diagram we try to find all necessary changes to our model to realize our goal. We start with the stakeholders, the business processes and services. Who are involved? Do we need to add new services? Which processes are involved, are there steps we need to add? Do the affected work processes need new or changed application services, which changes are needed in the applications layer? Are there changes to make in the infrastructural layer? We further identify services that will become obsolete at a certain point in time. In each cycle we add to our list of changes and note the dependencies involved. When all requirements are fulfilled our change list is complete and we can use the list to create the to-be model. But we also want a migration path. For that purpose we now[10] use a simple but clarifying diagram, a *dependency diagram* to visualize the dependencies and the impact of our changes as a preliminary step in designing a Paradigm migration model. Note that our change list is not final. Designing the migration can uncover that we need additional changes.

The business process in figure 5.6 also shows the design process the ArchiMate way. It is straight forward to recognize our state diagram. To keep the view in balance we aggregated some of the nodes and transitions into the process *Impact analysis*. In addition, we can also show the stakeholder and the

[10]We did identify the dependencies earlier, in the textual description and in the discussions, using trial and error.

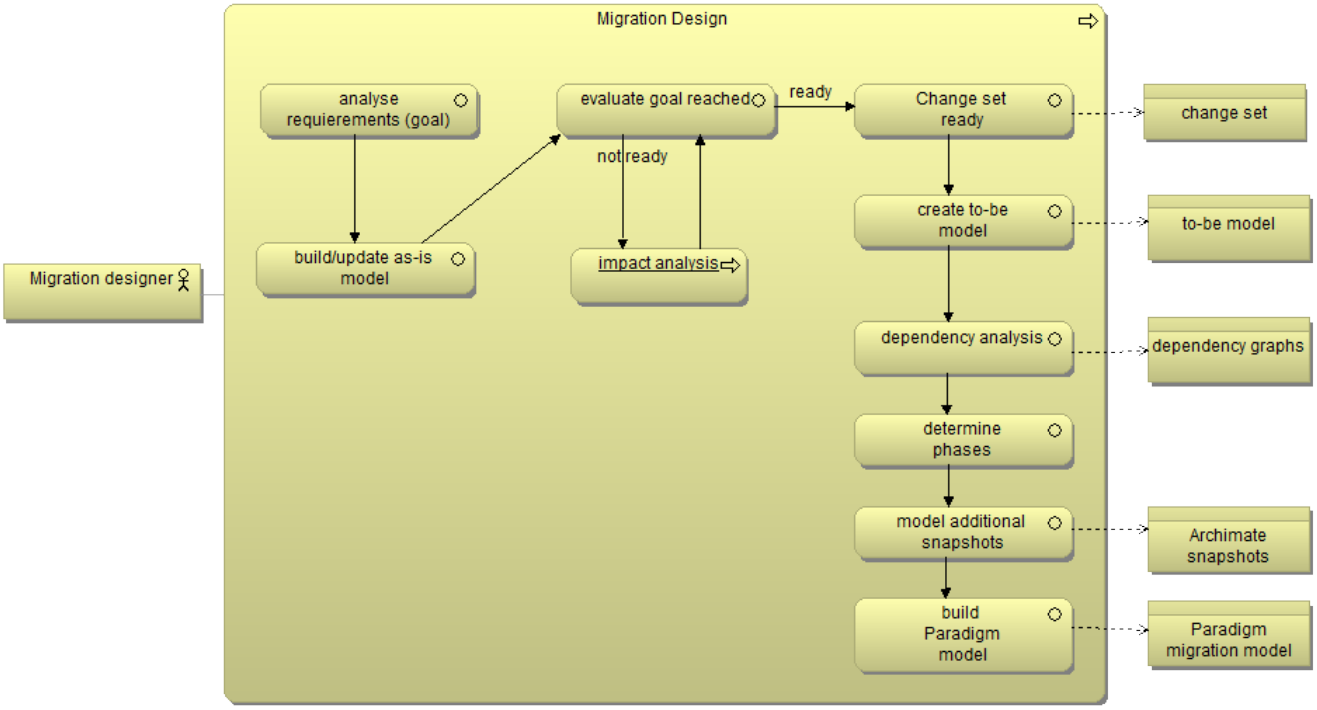


Figure 5.6: The migration design (ArchiMate Business process view)

results of the process, the change set and the models. We visualize that the *as is model* is changed by this process (or even created) and that the *to be model* is captured *inside* our current model.

This may seem trivial, as a new model can always be described as the current model with nodes and relations removed and nodes and relations added (to-be model is as-is model minus removals plus additions) but this only captures the static differences at the global level. The Paradigm model will visualize the dynamic aspects, the dependencies and the coordination of the changes at all levels.

But first we apply this method to our migration at hand. We summarize and label the changes, using the worker and the global perspective. This results in the following basic set. The set consists of migration actions, services and dependencies.

- The new authentication, print- and file services must be available. The change action needed is *switch vlan.* This provides the new services by virtually connecting the workstation to the UvT services. From our as-is model we can see that this switch has considerable impact, it invalidates the current services.
- The worker must retain the private network data, a *copy* of the *private data* is necessary, timed just before the (individual) migration.
- the worker must retain access to the shared network data, the *shared data.*
- the worker must be able to use the new services, he needs a new version of the operating system and client software, provided by the change *UvT image.*
- The new image needs to be configured (*configuring)* and the worker needs *instructions* to use the new services delivered by the changes *switch authentication* and *switch file system.* The first switch for the worker is prepared centrally, by creating a new account and invalidating the old account. The second switch, the worker using the new file system exclusively, has also been prepared at the central level.
- the old local data must be kept before the re-imaging takes place, using a *backup* action.

Analyzing these actions and changes using the as-is model we can see we need additional changes at the process, application and infrastructural level

- The current accounts have to be added to new authentication service, *with the action AddAccounts.*
- The workstations need to be registered in the new network environment, the action *RegisterWorkstations.*
- The local printers should be made accessible in the new environment, to be provided by the action *RegisterPrinters.*
- With a concentrated, all at one time migration, realizing access to the shared data is trivial. During the migration no access is possible, and after the migration the new shared data is available. If we are using a phased migration, write access to the shared data can be guaranteed by delaying the transfer to the new file server until all workstations are migrated by a new action *Providing the Original File service* in the new infrastructure[11]. This is a new possible constraint which will be resolved at a later stage.
- We need a provide a new *Workstation backup service,* an *Imaging service* and an action to *Switch* the *VLAN.*
- We need procedures and checklists to help perform the worker changes and the changes to the workstation, the action *Provide Work Documentation.*

Some of the infrastructural changes need further investigation. We need to try out the changes in a separate testing environment to analyze the impact. This will uncover some new dependencies and produces turn around times. Other infrastructural changes have a low impact and are not critical, because they can be realized as independent actions. They can be tested in the testing environment and performed in advance. The changes AddAccounts, RegisterWorkstations, registerPrinters will not be presented here.

We have to transform our change set into migration processes. At this point in time we could a project planning tool using Gantt Charts and Work Breakdown Structures. Then we can use tasks, predecessor relations between tasks and resource sharing constraints to model different possible orderings of the tasks and finding the critical path. By changing the constraints, and changing the resources available using the visual feedback of the diagrams and the calculated costs of material, time and work we can try to find a optimal solution. But this method is not sufficient for our goals. Our tasks depend not only on preceding tasks or resources but can create and *invalidate* services. This can not be modeled using the basic project planning tools.

To help us getting a grasp on the dependencies we will show our findings visually in a dependency diagram (see figure *5.7).* In the diagram we see the actions, the migration services and the relations between them, while concentrating on the worker perspective. We first make a diagram without the grouping, using the actions, services and relations between them. Then we add the grouping nodes *pre*, *post* and *new services* to summarize and phase the actions. For more complex migrations automated tools can be used to identify groups of connected actions. We use normal boxes to show phases and groupings, divided boxes for the actions and rounded boxes are used for the services. The edges depict the dependency on another service or action.

For now we assume that all actions take place at the local, worker level and that they are dependent on the presence of the worker. The nature of this dependency varies, for *instruction* it is clear that the worker needs to be present, for the backup, imaging and copying the connection the worker must have stopped working with the old workstation. This dependency will be illustrated more clearly later in the Paradigm models.

The boxes *post* and *pre* show the two phases or segments of the migration. The action boxes with the migration changes have an extra attribute, the turn around time.

Each arrow points in the direction of the dependency. A dependency can be an *need-or-availability* relation, e.g. the *new auth(orization)* service is only available after the action *switch authorization* is

[11] A separate dependency graph is not give here.

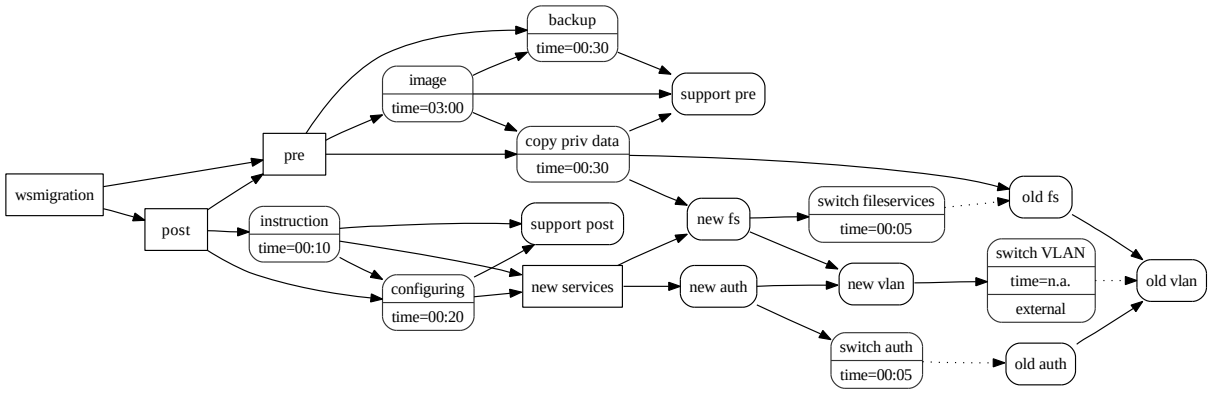


Figure 5.7: Dependency graph workstation migration (worker level)

performed. It can also be a *consist-off* relation, like the relation between *wsmigration* and *pre/post*. The *pre* segment is build from three preparing actions, the *copying of the private data*, the *backup* and the *imaging*. All three actions depend on the availability of the service *support (pre*). Note that the copying can only be done if both the old and the new file system are available. The *post segment* is dependent on the *pre segment* and through the actions *configuring* and *instruction* on the availability of the new *services* namely *new authorization* and the *new file services.* It is also dependent on the availability of the *support (post*). The new authorization is available after the action *switch authorization* but it also needs the *new vlan* just like the *new file system*. The old authorization, the VLAN switch and the old file system depend on the availability of the old VLAN. Each migration action can have a special kind of dependency, the *invalidates* relation, shown here as a dotted line. The *switch VLAN* invalidates the old vlan service and the switch file services invalidates the old file system. Note that the configuring and the instruction need the new services to be present. Using the diagram we can:

1. Identify which new services are needed and which infrastructural changes need to be present at the moment of an individual workstation migration.

2. See the possible orderings of the local migration actions. For each action and migration change we decide who will perform the action and at what level.

3. See, looking at the endpoints of the diagram that the resources or services *support pre, support post* and the *VLAN switch* are the most essential items.

4. Identify conflicts. To illustrate this we show the impact of VLAN switch in figure 5.8. It invalidates the service *old vlan*. We have shown this by coloring the nodes that are dependent on this node. We have to solve a problem: the copying of the private data is depending on both file services and consequently on both VLANs. At this point we have discovered that we need additional changes for the migration. In the next paragraph we will show a remedy using a new dependency diagram and we create a Paradigm model for the global migration process see figure 5.10.

Using this method we identify changes and show how the changes are grouped end related. We discover which actions have to be performed at a more global level due to a conflict situation and we can evaluate which actions must be performed at this level. Conflicting migration actions can be identified.

5.3 The migration process: the infrastructural preparation

In this simplified dependency diagram, in figure 5.9 we can see that switching the VLAN for the workstation would mean losing the old services. The gray dotted arrow indicates that we have to perform another infrastructural change, the *Provide Original Services* change. We can implement this on the workstation level or at the server level. For obvious practical reasons we choose the global level by adding network cards to the servers and connecting them to the new VLAN. This change integrates the earlier *Provide Original File service* constraint. We now show the coordination that is

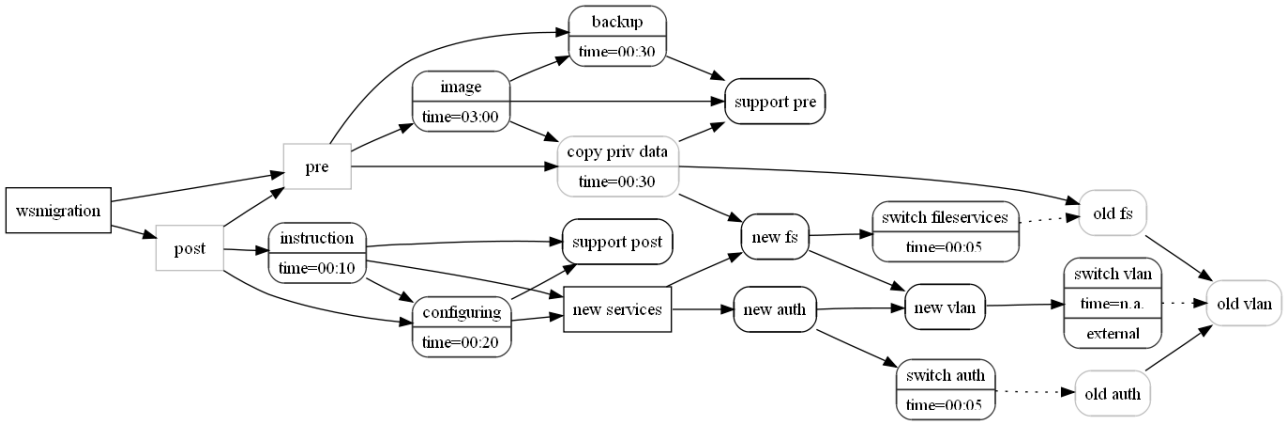


Figure 5.8: The impact of the VLAN switch

needed to implement this change. There are three roles involved, the infrastructural *Migration manager* is responsible for the infrastructural migration process, the *Network manager* represents the external party responsible for the UU network, capable to perform the changes to the network and the VLAN switch and the *Server manager* who implements the internal configuration changes. We show the three detailed Paradigm processes in figure 5.10, and the subprocesses in figure 5.11 and figure 5.12. The processes are connected with consistency rules given in table 2.

The Migration manager starts by requesting a change from - in arbitrary order - the Network manager and the Server manager. The two jobs can partly be performed in parallel, but the testing of Server manager can only be performed when the infrastructural changes is completed. This means that the Server manager has to know when the network change is ready. Then the migration process waits for the Network manager to signal that the job is done and after that waits for the Server manager to signal that the server changes are ready. We will model this communication using the subprocesses and global states. For the Network manager process we design three subprocesses see figure 5.11. This partition we call SD (service delivery). In the first subprocess on this level, *Free*, we are waiting for an outside request, that will come from the Migration manager. After receiving the request it will be installed and tested. This step corresponds with installing network equipment connecting the UvT and the UU network, realizing a new virtual LAN to connect to the FKT network and enabling this new VLAN on the network connection point (VAP) in the server room at the FKT. After that it will be reported that the work is done. It is interesting to note the differences with the detailed process. The subprocesses show how the outside world interacts (or can interact) with this process. The difference between the global transition form Install & test to Finish and the transition from Finish returning to Free is subtle. When starting In Finish the process is not yet listening to new requests. We can couple the process by choosing between the two 'signals'. The three states of the detailed process only model the internal process, without the coordination with the outside world. As we said earlier, this has the advantage that we separate the internal workflow from the interaction, keeping both compact.

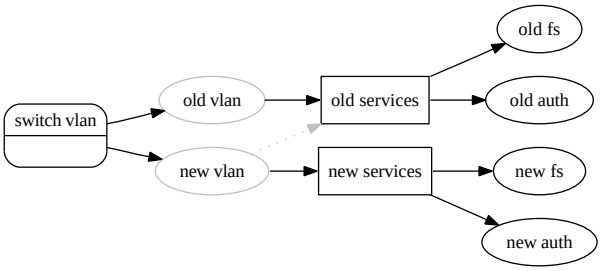


Figure 5.9: Dependency on VLAN

In the next figure, 5.12 we see the subprocesses and the global states of the Server Manager. We name this level of subprocesses Config. It is a partition of the detailed process. Like the Network manager we must be able to react on the request for change, coming from the Migration manager. After that we are in the subprocess (and global state) Install, in which the network cards are placed. But we can only leave this subprocess when we are notified that the Network manager is ready, which

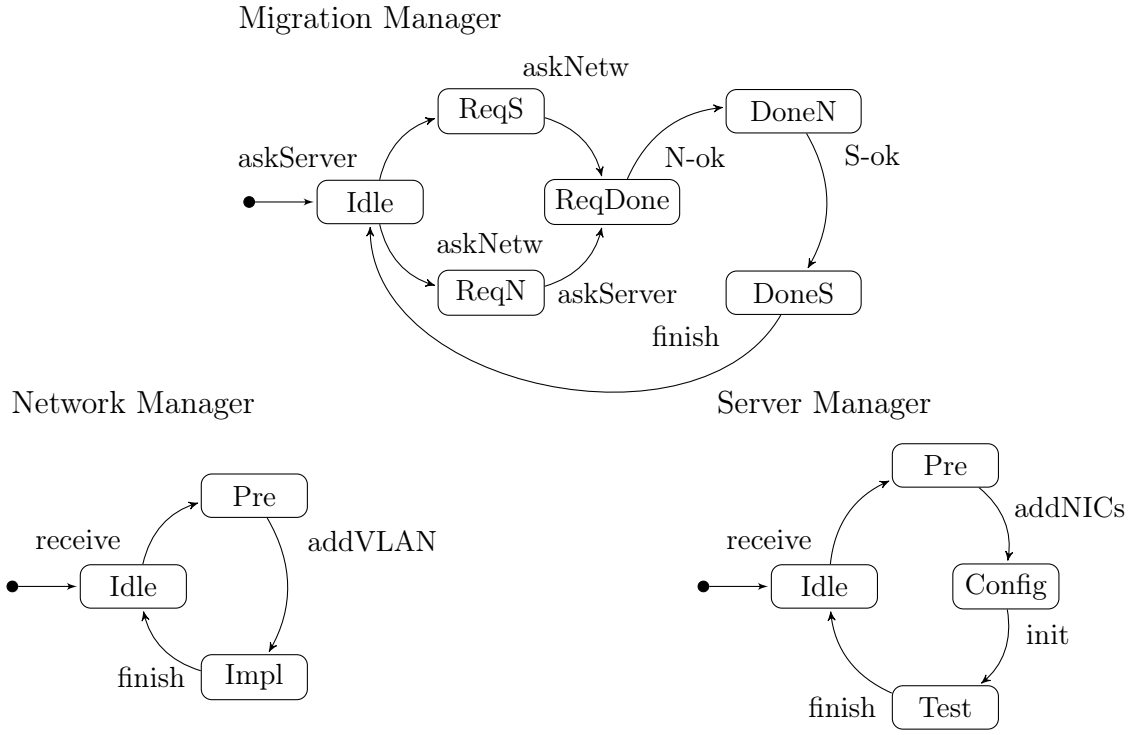


Figure 5.10: The infrastructural preparation*:* the detailed processes

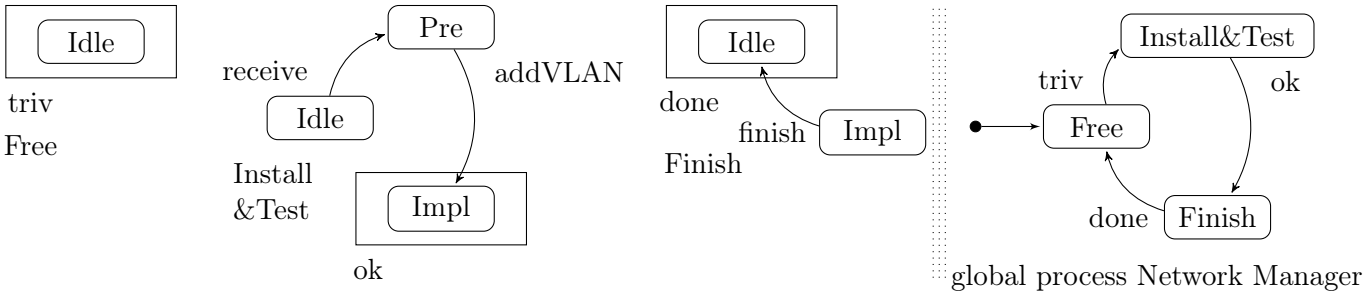


Figure 5.11: The infrastructural preparation*:* the partition SD and global Network Manager process

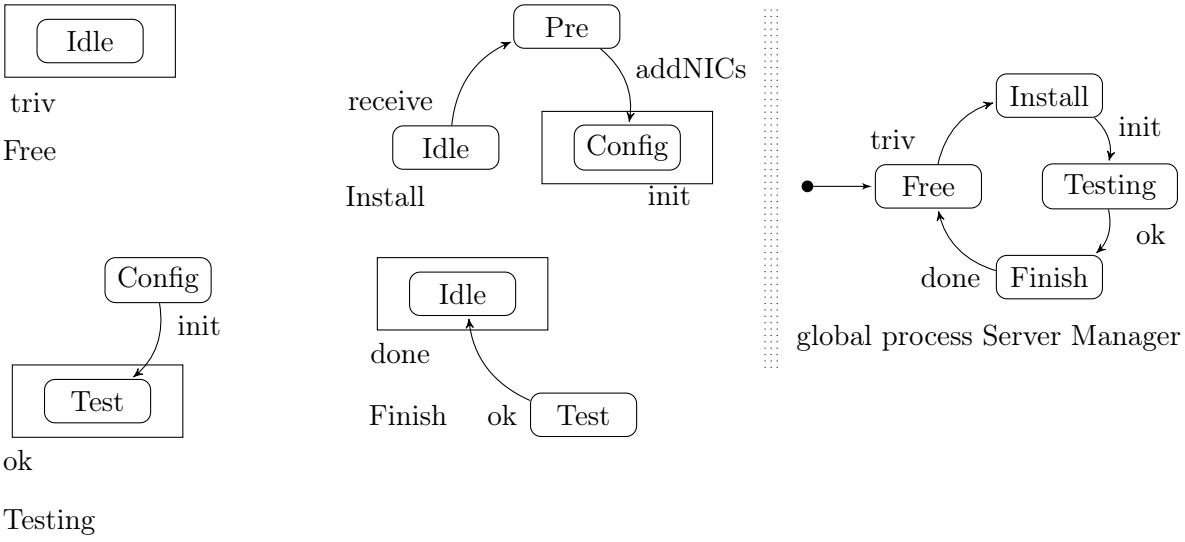


Figure 5.12: The infrastructural preparation*:* the partition CONFIG and global Server Manager process

enables the testing subprocess after connecting the new network cards to the new connection point. The testing is done inside the trap *ok* of NetworkManager (SD). The trap can only be left when the testing is done and this is acknowledged by the managing process i.e. the Migration manager. We now return to the global Free state.

Using Paradigm we can show the coupling between the (atomic) physical infrastructure change, the action taken by the external network infrastructure manager and how this is connected to the global migration. Rules **im1-3** describes the detailed actions taken by the Infrastructural manager to make a second VLAN and network outlets available. Rule **im3** also couples adding the network changes with the start of the testing in the process Servermanager. In this case the Infra manager has a managerial aspect and the Server manager a worker aspect. Another possibility is to model that the Migration manager, after receiving the Infra-is-ready signal would enable the Server manager to change from Install to Testing. Rules **sm1-4** show adding the physical network cards (NIC) to each server, and configuring and testing it, the detailed steps of the Server manager. The consistency rules **mm1-7** define the detailed steps of the Migration manager and connect them to starting the Infra manager process in **mm1** and **mm4** and the Server manager in **mm2** and **mm3.** The **mm5** rules states that Migration manager receives the signal that the Networkmanager is ready and at the same time signals the Server manager to start testing. The **mm6** rules receives of the signal that the Server manager has finished. Rule **mm7** is the detailed step of returning to the start position and letting the both the Network Manager and the Server manager know that the migration is ended..

We can also see the next phases of the migration. After the preliminary step the workstation migration will be performed, and some cleaning up can be done, the connections to the original network can be removed, services and servers no longer in use can be deactivated. But we are not ready for that yet. We don't have any rules yet to perform these steps. We can only perform the first migration step, to the global phase *Double services.* We still have to design the workstation migration, the second step.

In figure 5.13 we show the equivalent ArchiMate view, the Business Collaboration view. We can use the actor symbol to show who controls each process. We changed the process slightly. In the Paradigm model we can choose to request the VLAN change and the server change in arbitrary order. In the ArchiMate model this is simplified: first comes the VLAN request and than the server change, but this

im1	*Network manager*:	Idle	$\underrightarrow{receive}$	Pre	
im2	*Network manager*:	Pre	$\underrightarrow{addVLAN}$	Impl	*
	Server manager(CONFIG):	Install	$\underrightarrow{init}$	Testing	
im3	*Network manager*:	Impl	$\underrightarrow{finish}$	Idle	
sm1	*Server manager*:	Idle	$\underrightarrow{receive}$	Pre	
sm2	*Server manager*:	Pre	$\underrightarrow{addNICs}$	Config	
sm3	*Server manager*:	Config	$\underrightarrow{init}$	Test	
sm4	*Server manager*:	Test	$\underrightarrow{finish}$	Idle	
mm1	*Migration manager*:	Idle	$\underrightarrow{askServ}$	ReqS	*
	Network manager(SD):	Free	$\underrightarrow{triv}$	Install&Test	
mm2	*Migration manager*:	Idle	$\underrightarrow{askNetw}$	ReqN	*
	Server manager(CONFIG):	Free	$\underrightarrow{triv}$	Install	
mm3	*Migration manager*:	ReqS	$\underrightarrow{askNetw}$	ReqDone	*
	Server manager(CONFIG):	Free	$\underrightarrow{triv}$	Install	
mm4	*Migration manager*:	ReqN	$\underrightarrow{askInfra}$	ReqDone	*
	Network manager(SD):	Free	$\underrightarrow{triv}$	Install&Test	
mm5	*Migration manager*:	ReqDone	$\underrightarrow{N-ok}$	DoneN	*
	Network manager(SD):	Install & Test	$\underrightarrow{ok}$	Finish	,
	*Service manager(*CONFIG*)*	Install	$\underrightarrow{init}$	Testing	
mm6	*Migration manager*:	DoneN	$\underrightarrow{S-ok}$	DoneS	*
	Service manager(CONFIG):	Testing	$\underrightarrow{ok}$	Finish	
mm7	*Migration manager*:	DoneS	$\underrightarrow{finish}$	Idle	*
	Network manager(SD):	Finish	$\underrightarrow{done}$	Free	,
	Service manager(CONFIG):	Finish	$\underrightarrow{done}$	Free	

Table 2: the global migration process: the consistency rules Crs_{infra} for the infrastructural preparation

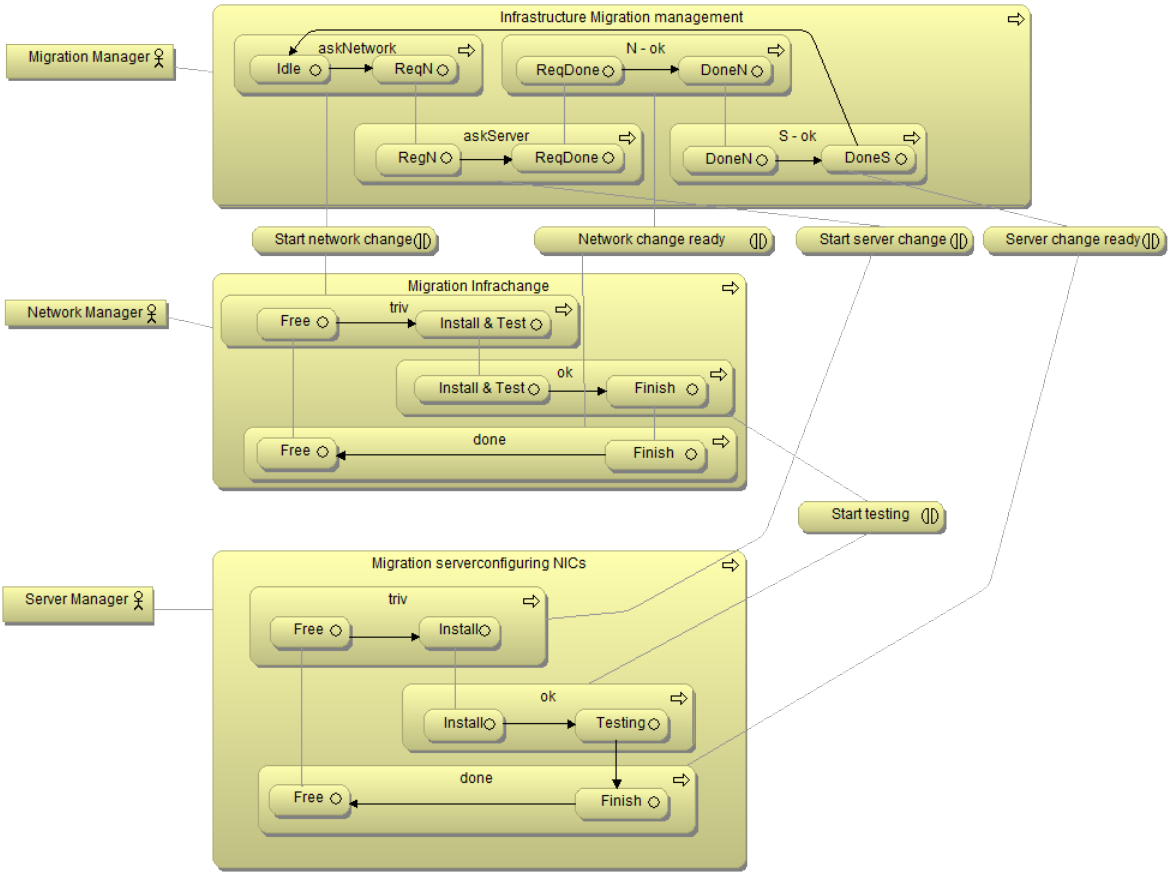


Figure 5.13: the global migration process: the infrastructural preparation (ArchiMate business collaboration view)

is not essential. The collaboration items are used to show the coordination. Unfortunately there is a mismatch. In Paradigm a coordination connects two or more *transitions* in different processes. But the ArchiMate collaboration used to depict the coordination connects *processes or activities*. One solution to circumvent this, is to connect the processes and place the endpoint near the transition/trigger. An alternative solution is to add a grouping business process when there are more than two activities in a process. When there are two states it is clear to which transition the coordination refers. In some cases we have to add a copy of a state to realize this. To express that the Paradigm coordination connects transitions we name this additional business process after the internal transition.

We have now described the actions that change the As is model into the first snapshot *Double Services*. The Paradigm rules which document the models are also sufficient to build the new business processes in the ArchiMate model. If we could express the other changes, like the actual infrastructural changes in Paradigm-like rules, we have a complete system to document any coordinated set of changes in a ArchiMate model. For example the change *addNICs* in the detailed Server manager process i*s* equivalent with adding an assign relation in the infrastructural layer of the ArchiMate model from each server to the *network UU Lichtpad.* Stretching the Paradigm notation we could formulate the following rules in table 3, documenting the infrastructural changes between our As Is model (see 4.2) and the first snapshot Double services(4.4). In set notation $CR_{DoubleServices} := CR_{AsIs} \cup CR_{addNetwork}$. Note that we do not give a complete description of CR_{AsIs}here. $CR_{support}$ is a small subset.

The first rule **im** gives the *addVLAN* coordination. The second VLAN with the Lichtpad connec-

im	*Network manager*	Pre	$\underrightarrow{addVLAN}$	Impl	*
i1		network UU Lichtpad<network>	$\underrightarrow{associate}$	UU VLAN LP<network>	
i2		UU VLAN LP<network>	$\underrightarrow{associate}$	UU internal<path>	
i3		UU internal<path>	$\underrightarrow{associate}$	Lichtpad UU-UvT<path>	
i4		Lichtpad UU-UvT<path>	$\underrightarrow{associate}$	UvT network<network>	
m1	*Server manager*	Pre	$\underrightarrow{addNICs}$	Config	*
i4		dienaar02<device>	$\underrightarrow{assign}$	network UU Lichtpad<network>	
i5		dienaar04<device>	$\underrightarrow{assign}$	network UU Lichtpad<network>	
i6		dienaar05<device>	$\underrightarrow{assign}$	network UU Lichtpad<network>	

Table 3: $CR_{addNetwork}$, the infrastructural changes in Paradigm notation, going from *As Is* to *Double Services*

tion is realized by the rules **i1-i4** describing adding the nodes and edges which connect the Utrecht infrastructure to the UvT infrastructure using the Lichtpad communication path. Between brackets the ArchiMate type of the node is given, and the name of the edge is the ArchiMate type of the connector. **m1** describes the expansion of the action *addNICs* (see the previous table) into the atomic infrastructural changes **i4-i6**, which describe connecting the servers to the new *network* UU Lichtpad.

In table 3 the coupling sign * can be read as 'is realized on the infrastructural level by'. We add elements to the model, reflecting the additions in the real world. Using this notation we have a powerful way to express coordinated atomic changes in an ArchiMate model.

5.4 Designing the workstation migration

After the infrastructural changes we are now in the phase *Double Services*. We will return to our workstation migration. A migration process could have been constructed from scratch, but using an analogy to a well known existing process reduces the possibility of alignment problems. From the worker perspective the infrastructural migration is just a special case of an IT support incident. For the scheduler and staff roles the workstation migration is more complex, but the structure is similar.

5.4.1 Individual workstation migration (all actions at worker level)

We start with a simple approach. All migration actions will be performed in the worker-support process. Interestingly the only addition we really have to make to our support model is renaming the worker

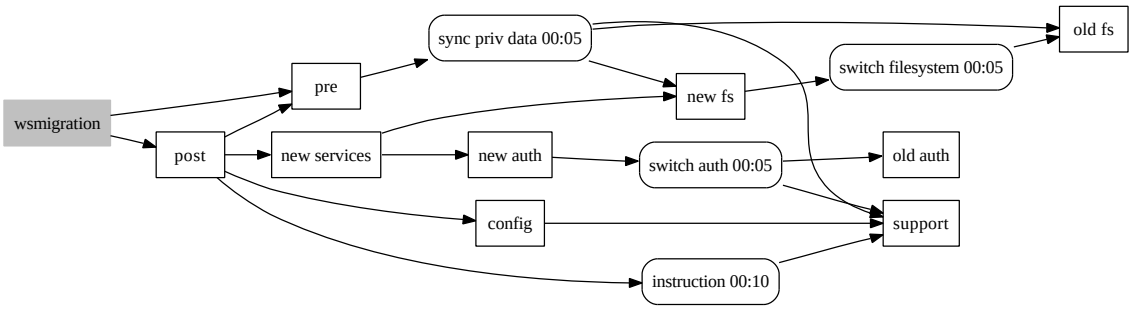


Figure 5.14: Dependency graph Workstation migration, second version (worker level)

signal *ask(support)* to *ready-to-migrate.* At this time we could use a special Paradigm adaptation component *McPal,* to be explained later to transform the process, but this first model is only a thought experiment. We delay the use of McPal until we have a definite migration model. We just change the model and the consistency rules [12]We now have a (suboptimal) workstation migration model.

In the migration phase *Double services* (with the new infrastructural services available but not yet in use) the employees work individually on their workstation and no coordination is necessary. The next action is the actual workstation migration. Each employee can signal *ready-to-migrate* which is picked up by the (migration) Scheduler. The critical section is the work that needs to be done by a member of the staff team (backup and transform the old station into a new UvT workstation, configuring and instructing the user). After the switch the worker process continues in the migrated situation and returns to the normal support situation. The team is ready to react on the next message *ready-to-migrate.* The *wsmigrated* phase is reached after all workers/workstations have been migrated. The migration actions that have to be taken by the support staff 'fit' in the Staff state *Support.* We could add the migration actions to the Staff process inside the Support state, making it a superstate, but this is not a fundamental change. After the migration the migration states are removed and the original support request is restored.

If we analyze the result using the data from the dependency diagram of figure 5.7 we conclude that in this simple approach all actions are performed by the same staff member at the worker location. The worker has to wait until his data is copied and the workstation is backed up and transformed. This will lead to unacceptable total turnover times.

To come to this conclusion we obviously do not really need Paradigm, but Paradigm forces us to separate and visualize the Worker and Support processes making it possible to reach this conclusion earlier.

5.4.2 Individual workstation migration (second, more parallel version)

When we use the Worker perspective we realize that the worker needs an new workstation, but not necessarily his own workstation. We can further reduce the turnover time by doing the bulk of the copy earlier. We provide a new global service to *synchronize* the private data with the new private storage on the new file server. This service runs every night for all users not migrated yet. Now we only have to *update* the private data by *data syncing* during the individual migration, which only take a few minutes.

We create a new staff role *Workshop* responsible for the actions *backup* and *image* in the background. The Worker now doesn't have to wait for a backup and transformation of his own workstation: he receives a new (or refurbished) workstation prepared in advance by the Workshop. The Workshop will backup and refurbish the old machine in a separate process.

This simplifies our dependency graph at the worker level, see figure 5.14. We have taken out the obsolete references to the old and new VLAN and changed the copy private data action into a

[12]We have to decide what to do with the open requests. For now we assume a change in the Service desk work flow ignoring the old requests in the list R. We further assume all old Support assignments have been be carried out.

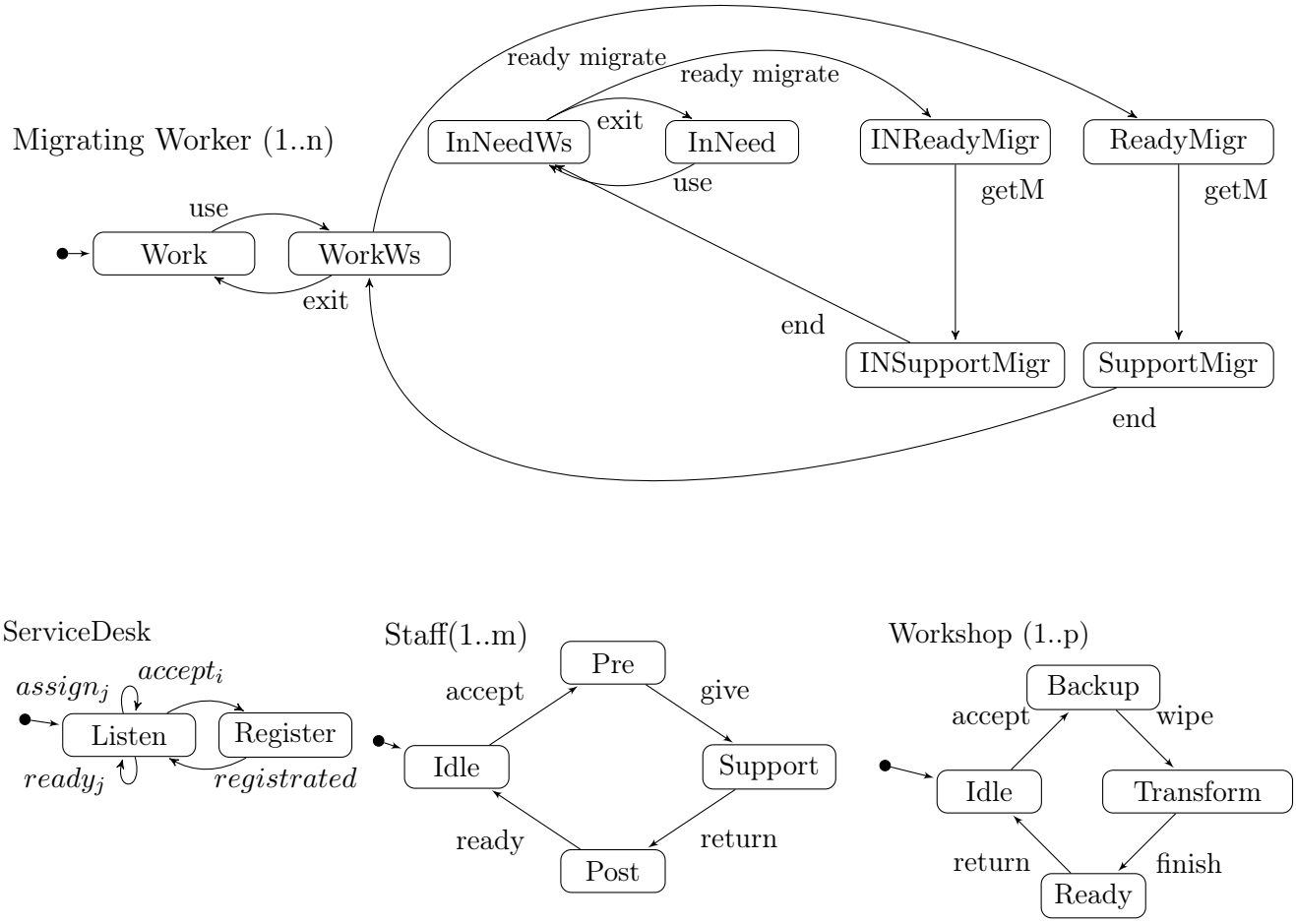


Figure 5.15: The detailed Migratingworker, Servicedesk, Staff and WorkshopStaff processes during the workstation migration

synchronization action. We can now create the new set of workstation migration processes see figure 5.15.

In the migration phase *Double services* (with the new infrastructural services available but not yet in use) the employees work individually on their workstation and no coordination is necessary. The next action is starting the actual workstation migration. The Migrating Worker is based on the normal Worker, the normal support ceases to exist and new requests can not be made. Instead each employee can signal *ready-to-migrate* which is picked up by the migration scheduler (human/software). There are two new states INReadyMigr and ReadyMigr. The first captures that there still is an outstanding normal support call. The Support state is supplemented by two Support states, INSupportMigr and SupportMigr. Again the first one remembers that there is an open support call. The subprocesses and traps can be found in figure 5.16. The goal of subprocess Standalone is to give the signal *requested*, when accepted by the Servicedesk Waiting is entered. The actual migration takes place during the last subprocess Migration Support which is started by the assigned Staff member.
The critical section is now the work that needs to be done by a member of the migration team (switching the old station for a new UvT workstation, configuring and instructing the user). For this process we can reuse the Staff model without changes. We do need a third type of component, Workshop, similar to Staff, with the states Idle, Backup, Transform and Ready. Like the Staff it has two subprocesses,

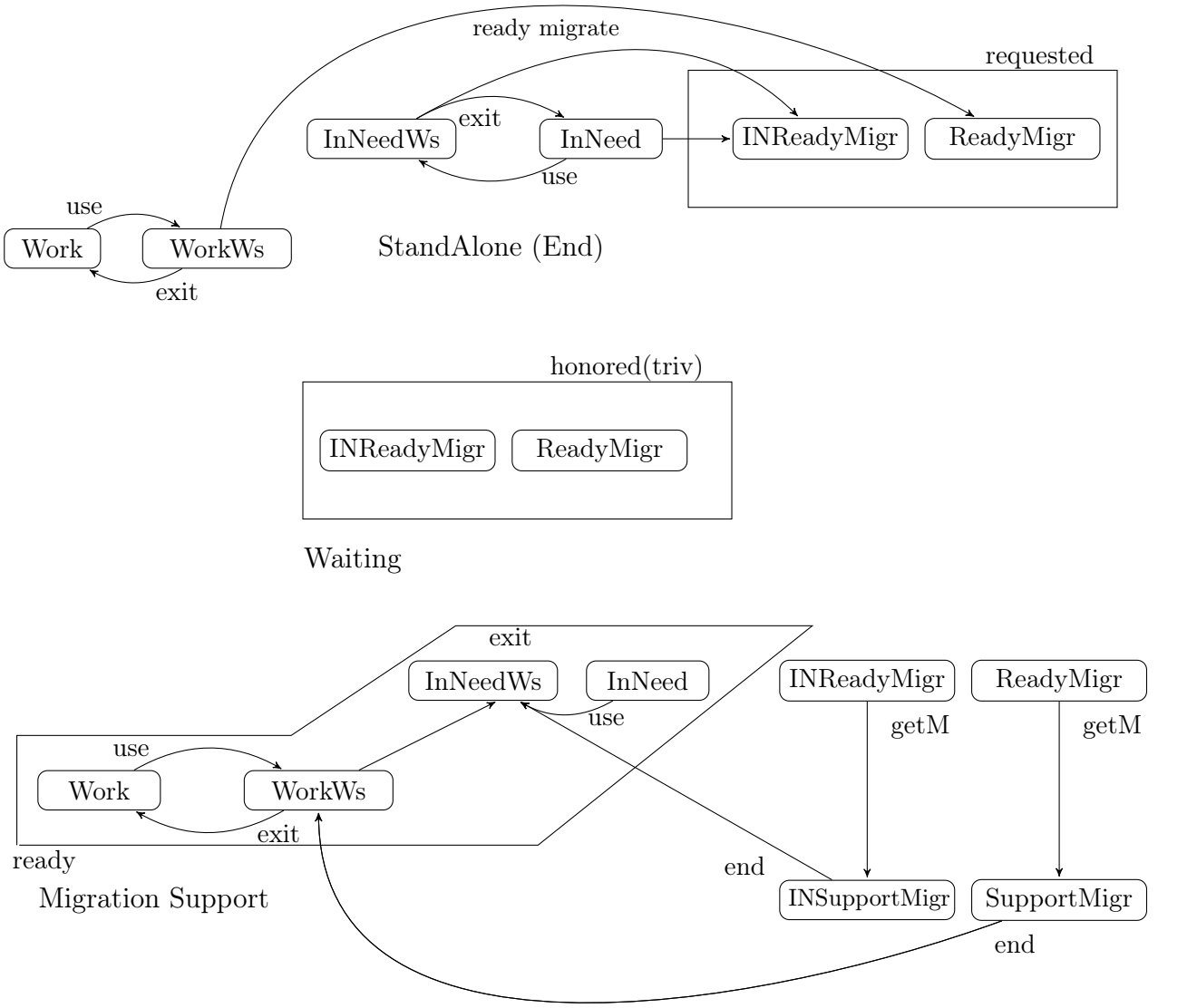


Figure 5.16: the subprocesses and global states of the Migrating worker

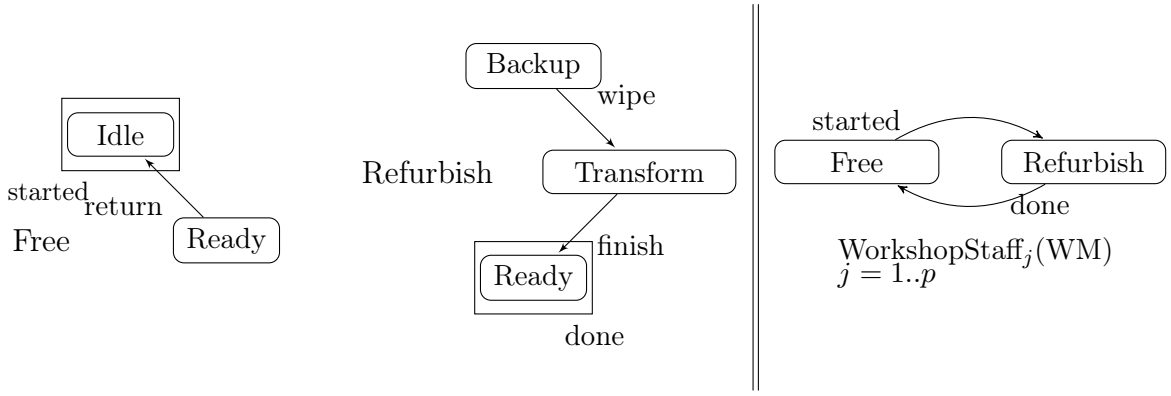


Figure 5.17: The subprocesses and global states of WorkshopStaff

wss1	*WorkshopStaff*$_j$:	Idle	$\xrightarrow{accept}$	Backup	
wss2	*WorkshopStaff*$_j$:	Backup	$\xrightarrow{wipe}$	Transform	
wss3	*WorkshopStaff*$_j$:	Transform	$finish$	Ready	
wss4	*WorkshopStaff*$_j$:	Ready	$\xrightarrow{return}$	Idle	
msd1	ServiceDesk:	Listen	$\xrightarrow{assign_j}$	Listen	*
	Migrating worker(MM):	Idle	$\xrightarrow{requested}$	Waiting	,
	Staff$_j$(SM):	Free	$\xrightarrow{accept}$	InSupport	,
	WorkshopStaff:	Refurbish	$\xrightarrow{done}$	Free	
	ServiceDesk:	[mcount:=mcount+1]			
msd2	Staff$_j$:	pre	$\xrightarrow{give}$	Support	*
	Migrating worker(MM):	Waiting	$\xrightarrow{honored}$	MigratingSupport	,
	WorkshopStaff:	Free	$\xrightarrow{started}$	Refurbish	,
msd3	ServiceDesk:	Listen	$\xrightarrow{return_j}$	Listen	*
	Staff$_j$(SM):	InSupport	$\xrightarrow{done}$	Free	
	Migrating worker(MM):	MigrationSupport	$\xrightarrow{ready}$	StandAlone (End)	

Table 4: Excerpt from the coordination rules of Crs$_{migrationsupport}$

shown in figure 5.17.

After the switch the worker process continues in the migrated situation and returns to a normal but unsupported situation. The migration team is ready to react on the next message *ready-to-migrate.* On every *requested* signal there are two parallel and synchronized processes, the support and, in the background, the backup of the switched machine is made and and the machine is transformed into an UvT workstation, which can be used in a next migration interaction. The *wsmigrated* situation is reached after all workers/workstations have been migrated. The migration team is in idle mode for an possible next migration. At the end of this phase the support scheduler will be restored and support resurrected.

The coordination can again be captured in the three strict rules of figure 4. Rule msd1 (cf. sd2 from the table 1) controls the coordinated start of three components, making sure there is a staff member assigned and a swap-machine available and delegates the start of the physical migration to the staff member, who in rule msd2 performs the migration support and reports back to the ServiceDesk.

When the individual migration ends (msd3), the old workstation is delivered to the workshop to start the backup and to be refurbished. A count of the migrated workstations is kept in the variable *mcount*. Note that Staff has a managing role in one rule of this coordination and a managed role in another.

Our task is to (1) temporarily transform the worker process into a migrating worker process and

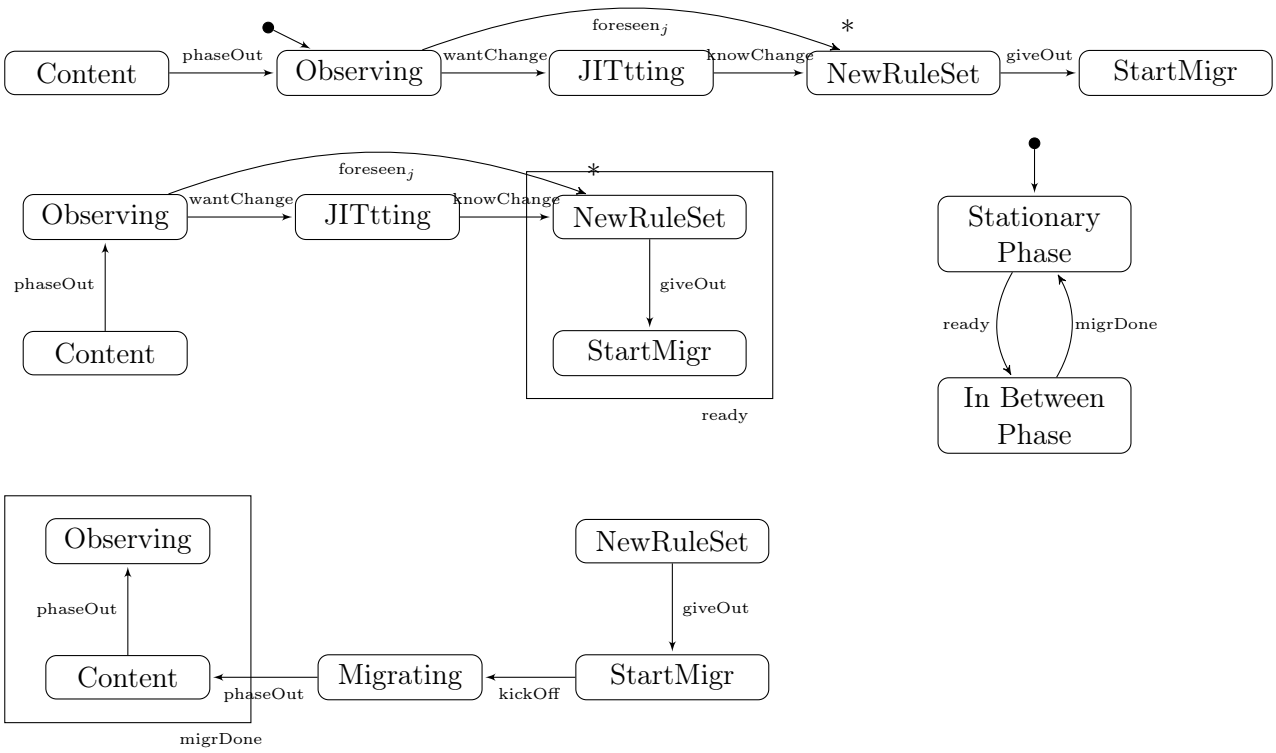


Figure 5.18: McPal, a standard component to manage change: detailed process and subprocesses of *Evol* partition

(2) make sure the Service Desk process is able to handle the workstation migration, (3) activate new workshop processes.

5.5 Coordinated transforming from Worker Support processes to Worker Migration processes and back

We now describe how we can transform the worker support processes into the newly designed workstation migration processes. We could use a new managerial component, but instead we use the standard change-manager McPal (acronym for Managing change Processes at leisure) to start and coordinate this transformation, see figure 5.18 . This component performs foreseen and unforeseen adaptations by changing the set of current consistency rules, referred to by the variable CRS. This variable is altered using *change clauses*.

In rest McPal's state is Observing. At a certain time at entering NewRuleset McPal is ready to receive or activate a set of additional temporary of permanent rules using the clause $CRS{:=}$ $CRS \cup Crs_{migr} \cup Crs_{toBe}$. Crs_{migr} describes special behaviour during the transition and Crs_{toBe} the targeted behaviour. But after performing this clause the model can not yet perform any of new behaviour, because the constraints and subprocesses are unaltered. In the next state (StartMigrate) the subprocesses of rule set Crs_{migr} will be activated in a coordinated way. The last step strips the old and the migration behaviour with the clause $CRS{:=}Crs_{toBe}$.

For every support component to be changed we create *Evol* partitions, that phase the *evol*utionary change. And we create the new component WorkshopStaff. After the workstation migration is complete McPal starts a new migration, restoring the original support situation.

The transforming subprocesses are on a high level. We start with the worker process in figure 5.19. The first subprocess is the original supported worker process, the last is the migrating worker.

1	McPal:	Observing	$\xrightarrow{wantChange}$	JITTing	*
		DesignProcess(CrsM):Deliver	$\xrightarrow{giveCrs_j}$	WaitNext	
2	McPal:	JITTing	$\xrightarrow{knowChange}$	NewRuleset	*
3		McPal:	[$CRS =$	$CRS \cup Crs_{phase1-2} \cup Crs_{migrationsupport}$	

Table 5: Transforming Support to Migrating Support: the rules

1	McPal:	NewRuleSet	$\xrightarrow{start}$	StartMigr	*
		McPal(Migr):Phase1	$\xrightarrow{ready}$	Phase1-2	
2	McPal:	StartMigr	$\xrightarrow{kickoff}$	Content	*
		Worker(Evol):Phase1	$\xrightarrow{triv}$	Phase 1-2	
3	McPal:	Content	$\xrightarrow{phaseOut}$	Observing	*
		McPal(Migr):Phase 1-2	$\xrightarrow{migrDone}$	Phase 2	,
		Worker(Evol):	$\xrightarrow{migrationDone}$	Phase 2	
			McPal[$CRS =$	$Crs_{migrationsupport}$]	

Table 6: Crs_{ph1-2}Transforming Support to Migrating Support

In between we combine the two. When designing a process migration we make sure that all states not present in the *to be phase* can not be entered anymore and have a successor in the state set of the new phase. This guarantees that these states are eventually not active so that the migration will start. Even when certain states, in this case the states InNeedWs and InNeed, are present in the phase 2, we sometimes want them to be not active, in this case because the meaning of the states shifts from 'a support request is present' and implicitly 'the migration has not started' in phase 1 to 'a support request is present and the migration is ready' in phase 2.

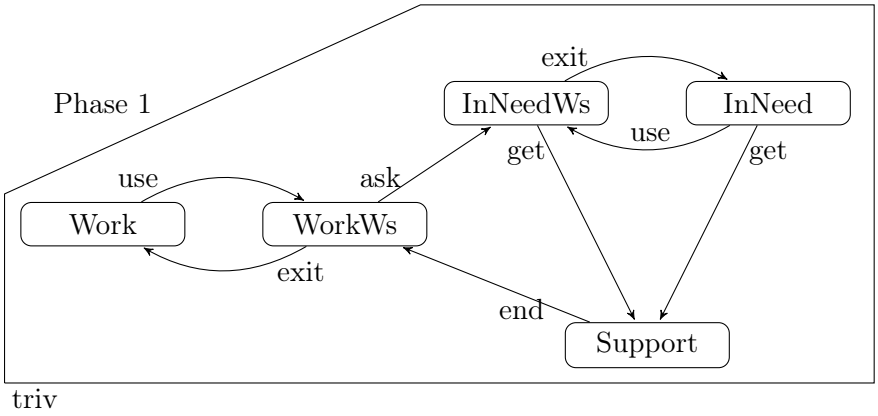
To go to MigrationSupport situation we first add new migrationsupport behaviour. We don't give the details here, we name this set $Crs_{phase1-2}$and then switch to the *to be* situation $Crs_{migrationSupport}$. This removes the old support-behaviour.

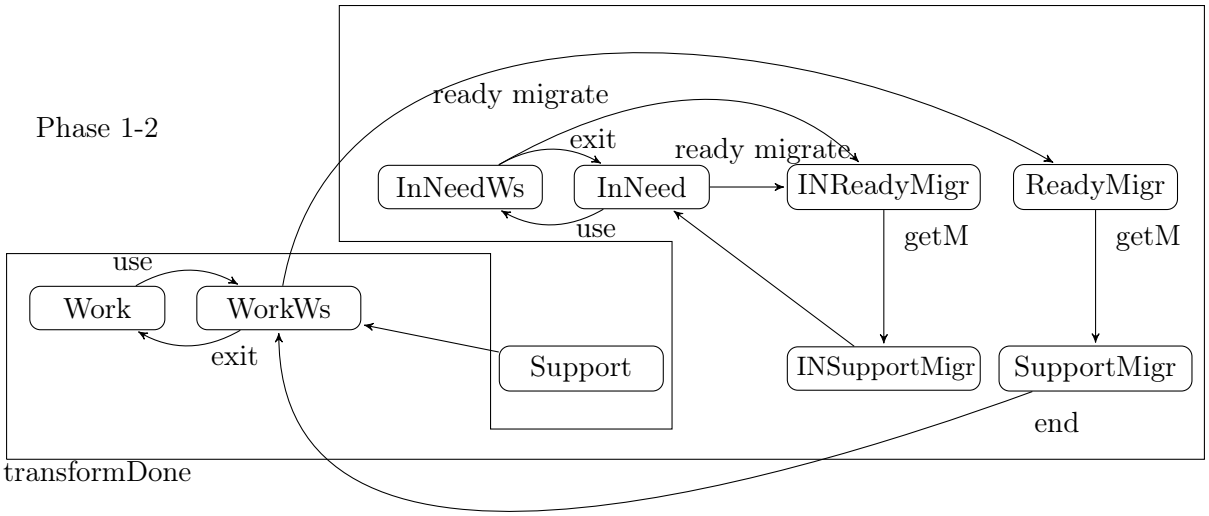
Assuming that all support is now finished, McPal now removes the *end* transition and introduces the new Workshop transitions and states $Crs_{workshop}$ and the new ServiceDesk rules CR_{MSD} in effect creating the Migration situation we designed in the previous section.

As a side remark: the rule sets should be seen as products of component Migration Design process in figure 5.5, McPal will start migrating when this process is ready. We will not model this process in detail here, in table 5we assume a subprocess Deliver and WaitNext and a partition CrsM which produces the rule sets *migrTo* aka *ph1-2, migrBack* aka *ph2-1* and *toBe* aka *migrationsupport.* McPal adds the first two rule sets to the current Support model. $Crs_{phase1-2}$ is detailed in table 6

The three rules of *Crs* add the start up of the McPal migration, first entering its subprocess *Phase1-2.* McPals next step makes sure the Worker is forced to start changing into a MigratingWorker. The ServiceDesk remains unchanged, at this time it is still performing the support routine. Only when in the next detailed step of McPal, where the worker gets pushed into the MigrationWorker situation and the new WorkShopStaff processes are active due to the CRS redefinition.
Each of the (Migration) Workers has now the opportunity to signal *ready_for_migration.* We assume all workers have give this signal. The number of migrations is registered in the variable mcount (see rule msd3). It is not modeled here, but at reaching the total number of workers ServiceDesk could enter a trap *AllMigrated* and doing so inform McPal. This would trigger a new migration, restoring the Support situation. See the first rule of table *8*

Note that all transformations are on-the-fly and preserve the coordination on the lower levels.





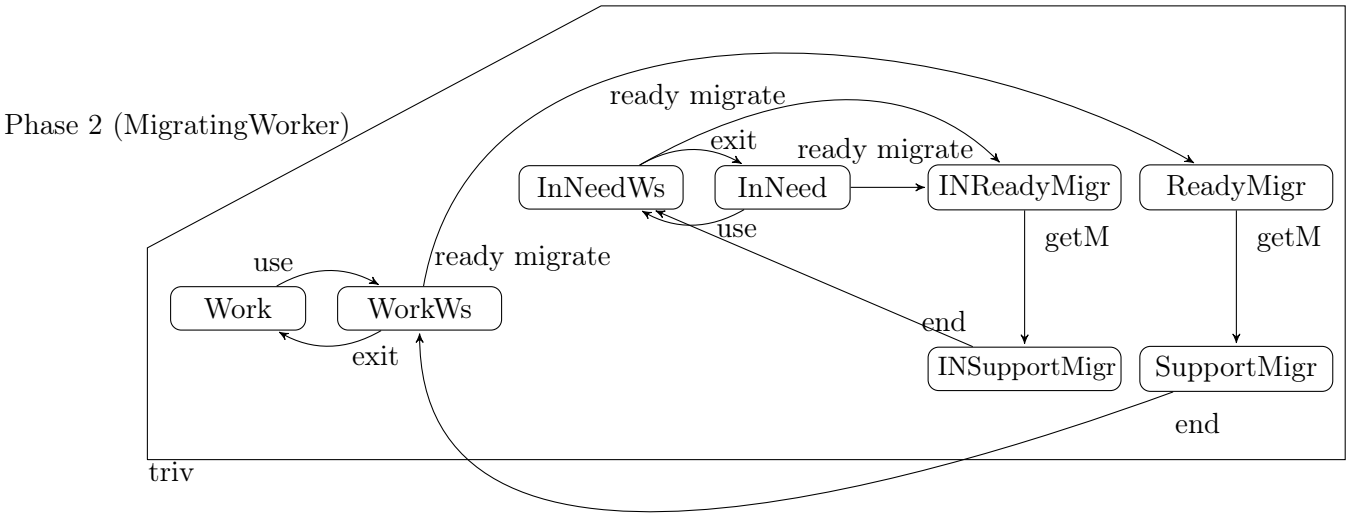


Figure 5.19: Transforming the Worker into Migrating worker, the Evol subprocesses.

1	McPal:	Observing	$\xrightarrow{wantChange}$	JITTing	*
		DesignProcess(CrsM):Deliver	$\xrightarrow{giveCrs_j}$	WaitNext	
2	McPal:	JITTing	$\xrightarrow{knowChange}$	NewRuleset	*
		McPal:	[$CRS =$	$CR\text{S}\cup Crs_{phase2-1}\cup Crs_{Support}$]	

Table 7: Crs_{ph2-1}Transforming Migration Support back to Support

1	McPal:	NewRuleSet	$\xrightarrow{start}$	StartMigr	*
		ServiceDesk(Evol):Servicing	$\xrightarrow{allMigrated}$	Servicing	,
		McPal(Evol):Phase2	$\xrightarrow{ready}$	Phase2-1	
2	McPal:	StartMigr	$\xrightarrow{kickoff}$	Content	*
		Worker(Evol):Phase2	$\xrightarrow{triv}$	Phase 2-1	
3	McPal:	Content	$\xrightarrow{phaseOut}$	Observing	*
		McPal(Evol):Phase 2-1	$\xrightarrow{migrDone}$	Phase 1	,
		Worker(Evol):	$\xrightarrow{migrationDone}$	Phase 1	,
			McPal:[$CRS =$	$Crs_{support}$]	

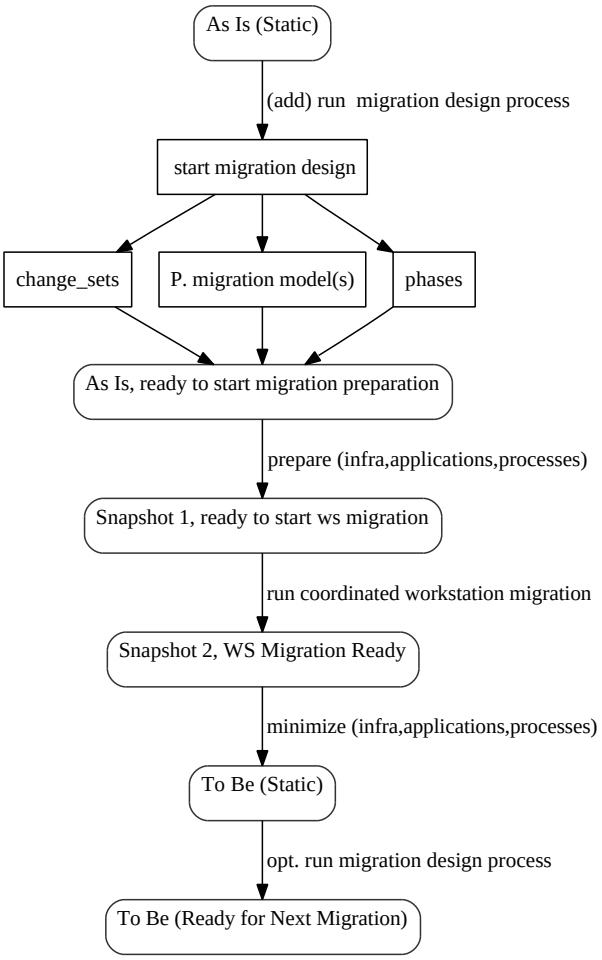
Table 8: Crs_{ph1-2}Transforming Support to Migrating Support

6 The complete picture

In the process of our work we have made a suite of Paradigm and ArchiMate models

1. We introduced ArchiMate and Paradigm (without McPal).
   (a) We described the infrastructural level of our case in ArchiMate, the As Is model and the other snapshots. The snapshots came from the actual business case.
   (b) Using Paradigm we created the original set of worker(support) processes with a scheduler(service desk) coordinating the support process.
   (c) We modeled the Migration design process which creates several migration processes (implementation of infrastructural changes, preparing the workstation migration, the actual workstation migration and finishing the workstation migration, performing finishing infrastructural changes)

2. We described the model transformations using the Paradigm McPal component (Observing->JITting->NewRuleSet) which
   (a) first transforms the original static As Is model into the As Is situation that is ready to start migration preparation,
      i. First McPal is at rest, no activity. McPal receives and deploys the new rule set ($Crs_{migrationdesigner}$) which describes and creates the Migration Designer as a Paradigm STD (and as an ArchiMate business process) (or alternatively starts the process Migration designer described by the rule set $Crs_{migrationdesigner}$) . This migration design process produces the change sets, the Paradigm migration models and the global migrations phases.



i. It produces a first set of necessary changes, the impact of the changes and the dependencies between the changes. After visualizing the impact and the dependencies in a diagram and further designing transforms them into coordinated Paradigm processes described by the rule sets $Crs_{infraMigration}$ and $Crs_{finishInfraMigration}$for the infrastructural changes and $Crs_{ph1-2}Crs_{migrationsupport}$ for the transformation of the support process into a workstation migration process and $Crs_{ph2-1}$to restore the original support situation. The level of the 'real' infrastructural changes, like adding a network card of changing a service has also been modeled using the Paradigm notation.

ii. McPal receives $Crs_{infraMigration}$ and at certain time (Event: Prepare infra migration) activates the infrastructural migration. The first phase-step in 2 gets active.

iii. Now the processes MigrationManager, InfraManager, and Local Migrator are added to the model. They model the requesting and monitoring of external and internal changes of the network infrastructure. This step is a common one in migration projects. In our case it is providing the new network services within the working infrastructure.

(b) signals that the As Is model is to be transformed into DoubleServices,

i. The three new processes perform in a coordinated way the infrastructural changes (addVLAN and addNIC) making the new services globally available.

ii. After finalizing McPal removes or deactivates the processes of InfraManager and Migrator. CR := CR / $\text{CR}_{finishInfraMigration}$

(c) starts the workstation migration comprising three internal phases, in 'DoubleServices',

i. McPal processes the rule set $Crs_{ph1-2}$and $Crs_{migrationsupport}$which prepares the workstation migration by transforming the worker and support processes into the workstation migration process (for each worker) while preserving the outstanding calls and starts all processes and adds/activates the Workshop process. (Phase Transform1-2)

ii. After this transformation the workstation migration can be started, coordinated by the Service Desk (Phase Migrating)

iii. After all workstation migrations are finished McPal, using Crs_{ph2-1} transforms the worker support processes into to the original processes, restoring the original calls, and removes the Workshop process. (Phase Transform 2-1)

(d) and starts the transition to the last phase, involving the minimizing of the obsolete services

(e) returns to the stationary situation, waiting for a possible new migration design

To summarize, we have structured the design of our migration in phases at different levels. In our case we have the global level of designing the *as is*, the *to be* situation and the migration phases (or snapshots) including the preparation of the transitions and the 'cleaning up' afterwards. We used ArchiMate to model the infrastructure, the applications and the processes of each phase. We have added migration processes to each phase, that can transform the phase into the next phase. The global design process produced intermediate building blocks for the migration. These migration processes change the elements of our model and can also create or transform other processes. We have used Paradigm to model the coordination and the constraints of the design and the migration processes and have integrated them in the ArchiMate models.

7 Observations

1. Paradigm modeling can capture the migration steps of an actual migration processes.

2. The migration design process can also be modeled using Paradigm, change sets and dependency graphs.

3. Paradigm can be used successfully to model the dependencies between infrastructural changes, application changes and process changes.

4. Paradigm can be used to model the coordinated changes to the model by mapping the ArchiMate nodes (the different types of *nodes*, application objects, services, processes and connections) to P. nodes and transitions, while preserving the type (for example in a UML stereotype)

It remains difficult to decide whether using the Paradigm approach in our real-life migration from the start would have been cost and time effective. Although the basic concept of Paradigm are clear and elegant, it takes quite some time to really get acquainted with Paradigm modeling. But building Paradigm models really forces (and helps) the modeler to more fully understand and communicate all the difficulties and dependencies of migration projects.

References

[1] M. Lankhorst, Ed., *Enterprise Architecture at Work, Modelling, Communication and Analysis*. Springer, 2005.

[2] H. Bosma, H. Jonkers, and M. Lankhorst, "Inleiding in de ArchiMate-taal," Telematica Instituut/ ArchiMate consortium, Tech. Rep., 2005, https://doc.telin.nl/dscgi/ds.py/Get/File-49772.

[3] "The open group." [Online]. Available: http://www.opengroup.org/

[4] S. Andova, L. P. J. Groenewegen, and E. P. de Vink, "Dynamic consistency in process algebra: From paradigm to acp," *Electron. Notes Theor. Comput. Sci.*, vol. 229, no. 2, pp. 3–20, 2009.

[5] L. Groenewegen, A. Stam, P. Toussaint, and E. de Vink, "Paradigm as organization-oriented coordination language," in *Proc. CoOrg 2005*, L. v. d. Torre and G. Boella, Eds. ENTCS 150(3), 2005, pp. 93–113.

[6] L. Groenewegen, J. Stafleu, A. Stam, and E. de Vink, "Paradigm and on-the-fly migration as constraint orchestration," 2006, wordt gepubliceerd.

[7] L. Groenewegen and E. d. Vink, "Evolution-on-the-fly with Paradigm," in *Proc. Coordination 2006*, P. Ciancarini and H. Wiklicky, Eds. LNCS 4038, 2006, pp. 97–112.