ALLOW

Adaptable Pervasive Flows

213339

Deliverable D4.3

Specification of adaptive algorithms for distributed control of adaptive flow control engine (Phase 2)

Contributor	USTUTT
Author	IPVS
Reference Number	ALLOW.USTUTT.D43.2011-07-04
Version	1.0 - Final
Date	2011-07-30
Classification	Public
Circulation	Project Team
Contract Start Date	2008-02-01 - Duration: 42 Months
Project Coordinator	USTUTT
Project Partner	USTUTT (IPVS), USTUTT (IAAS), UNI PASSAU, FBK, Imperial, ULANC

FET - Future and Emerging Technologies

Project funded by the European Community
under the FP7-Programme (2007 - 2013)

Copyright

Document History

Version	Issue Date	Author	Content and Changes
0.1	04.07.11	Klaus Herrmann	Coarse Structure
0.2	19.07.11	Klaus Herrmann	First complete draft version
0.3	25.07.11	Stefan Föll, Hannes Wolf	Second draft
1.0	30.07.11	Klaus Herrmann	Final version

Executive Summary

In this document, we describe the work conducted in WP4 over the final 18 months of the project. This work was primarily concerned with three areas of research: (1) flow distribution, (2) robust flow navigation, and (3) context prediction. It builds on the work described in Deliverables D4.1 and D4.2. D4.1 reported on architectural concerns with respect to flow control, and D4.2 reported on fundamental modeling and algorithmic aspects. The work reported in this document represents a considerable extension of the prior work. In the area of flow distribution, we present two algorithms that optimize a flow-based application with respect to user-perceived latency and energy consumption. In addition to this, we have investigated two associated fields.

Robust flow navigation is concerned with the question as how flows can be controlled robustly in the face of uncertainty and noisiness of context data as it is commonly encountered in the real world. We present mechanisms for dealing with this uncertainty and noisiness, both for single events and for event sequences. We show that the robustness of flows can be improved considerably over the case where no measures are taken. Note that in today's workflow systems, input data is usually assumed to be correct, which is not the case with pervasive flows that take their input data from sensors in their environment.

Context prediction is a fundamental enabling technology for proactive systems. We will show that flows and flow-like models allow for more accurate context prediction by exploiting knowledge about the temporal structure of an application. This temporal structure is inherent to flows and can be accessed very easily, unlike in other programming paradigms. We demonstrate that by associating context information with activity information taken from a flow, we can increase the accuracy of context predictions. Furthermore, we present a new way of generating such predictions that is based on model-checking concepts. This has not been done before and represents a major advance in context prediction research as it allows for much more expressive *temporal queries*. That is, a querying application or user gets information about when a specific context change actually happens in real time. Previous systems where limited to a time-discrete model and, thus, could only make predictions about the next context change without any information as to when this change would happen.

We cover each of these three major areas in a chapter. In each of the chapters, we will present in-depth evaluation studies and quantitative results that are based on the real-world case studies conducted in the project in the final reporting period.

Contents

List of Figures

Chapter 1

Introduction

In this document, we will describe the work that was conducted in terms of adaptive distributed flow control over the final year of the project. While the deliverables D4.1 and D4.2 mainly discussed *flow distribution*, we have broadened our view on flow control considerable over the final year and included a significant body of work in the areas of *robust flow navigation* and *context prediction*. These are two important aspects of flow control that have emerged during the project and that we have tackled beyond the work described in the Description of Work. In this deliverable, we will give an overview of this body of work in the area of adaptive flow control before we give detailed insight in each of the three areas including detailed evaluation results.

1.1 Overview of the Work Conducted

The area of flow distribution is concerned with the problem of finding and maintaining an effective and efficient distribution of the activities contained within a flow over the available computing resources. The basic idea of distributing flows is driven by the fact that flows need to maintain rich communication with their human users *and* with diverse services in their environment. Since the human users we regard are mobile, the available computing resources and the network connecting them it is not known a priori. Moreover, network bandwidth, stability and quality may be very diverse. Hence, effective mechanisms are required for placing different parts of a flow on those computing resources where they may be executed with the highest utility. We have investigated two possible *utility metrics* in this project. The first one is *user-perceived latency* [38]. This is basically the accumulated time that a user has to wait during the interaction with his flow. To allow users to operate unobstructed by any network effects (low bandwidth, high delay, etc) the activities of his flow have to be placed accordingly such that high-volume communication does not flow over low-quality network links. The second utility metric is *energy* [27]. In the common scenarios regarded in the project, mobile users are equipped with mobile devices (e.g. off-the-shelf smart phones) that are part of the computing resources used for executing flows. The rest of this computing infrastructure are backend infrastructure servers. The basic question that we investigated is: *In which way must a given flow be distributed in this computing infrastructure such that the limited battery capacity of the participating mobile devices is best preserved?* What would be an adequate algorithm for achieving such an energy-efficient distribution of flows? Preserving the energy of mobile devices is of primary importance since more energy consumption leads to shorter re-charge cycles, which in turn leads to more disturbances in the work process, e.g. requiring users to re-charge their mobile device in the middle of a shift. On the other hand, if the energy consumption is minimized to a known average value, cheaper devices with adequate battery capacity can be used, avoiding over-provisioning in terms of energy. Both effects lead to monetary benefits over the long run.

Flow navigation, i.e. algorithms for deciding how to navigate through the flow's transitions based on the available input data, is a central part of flow control. The flows in the ALLOW project navigate mainly based on context data collected from the real world. Therefore, real-world effects that occur when

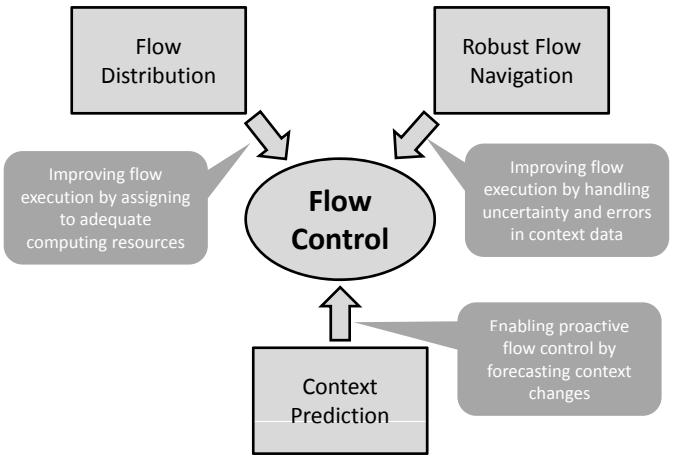


Figure 1.1: Flow control overview

context information is captured by sensing devices and categorized by context management and activity recognition systems can become a serious problem. We investigated mechanisms for ensuring *robust flow navigation* under such effects [79, 77]. Most of the context-aware systems proposed in literature assume that the information provided by these systems is correct. However, noise, cheap sensors, and classification errors produce data that can carry a high degree of uncertainty. Moreover, data can get lost due to hardware and software malfunction or simply because the raw data was too noisy to be categorized (*false negative*). The same software may find patterns of known events while such an event actually never happened (*false positives*). Finally, the order of events may not be recognized correctly due to timing problems and race conditions (*order errors*). All of these effects happen regularly in real systems, and a flow that navigates based on context data will fail very quickly under these conditions.

With *FlowCon* [79], *FlexCon* and *FeVA* [77], we have developed a set of mechanisms that exploit the *knowledge encoded within a flow* in order to correct these effects before the context data is actually given to the flow. *Flow knowledge* is basically the knowledge about the temporal behavior of the flow that is encoded in the order of activities and in the transition between these activities. In a *normal* (non-flow-based) application, the temporal behavior is usually hidden in the program code, and it can be extremely difficult to infer what the application will do at a specific moment in time, even if one has full access to the code. In a flow-based application, this temporal behavior is explicitly defined in the transition system. The individual activities, the transitions and the conditions under which the transitions are taken are open and accessible. Therefore, it is much easier to infer the past, current and even the future behavior from the flow. We exploit this knowledge to reduce the uncertainty of incoming context data and to repair errors stemming from false positives, false negatives, and order errors.

Finally, navigating a flow based on current and past context information may be sufficient for many applications. However, if the flow has to be adapted in *real time* (i.e. while the user is waiting) to work around some problem (e.g. a missing resource), and if such problems may occur frequently (due to the mobility and dynamic of the entire system) then having reliable information about *future context changes* is vital for flow control. It enables the system to be proactive by recognizing and circumventing possible problems before they actually take effect. Therefore, we have also investigated the area of *flow-based context prediction* [28, 29]. The basic idea of flow-based context prediction is to exploit the flow knowledge (see above) and associate it with context information to be able to foresee future context changes more accurately. We have proposed the *PreCon* system [29] that takes activity information (from a flow or some other source) and learns how these activities are connected with context changes. The result is a system that can learn the behavior of context information more precisely than existing systems.

Chapter 2

Flow Distribution

In a classical business context, workflows are executed on some powerful back-end server, and there is little need to think about distributing them. In a pervasive scenario where the users of the flows are mobile and may access them from different locations and with different network connections, this changes. A single back-end server may be hard to reach from some locations. For some parts of the flow, the data that needs to be communicated may overwhelm the network resources. Users may have to wait for prolonged periods of time for their data to arrive at their mobile devices. Finally, the mobile devices of users only have a limited energy budget. Constant communication is a big factor in the energy consumption, such that batteries may be drained quickly, lowering the usefulness of the entire system. These problems can be solved through flow distribution. That is, by fragmenting a flow and placing the various parts on specific computing nodes (infrastructure or mobile devices), certain key parameters concerning the usability, the effectiveness and the efficiency of the flow-based application can be optimized.

We have looked at two main concerns that necessitate adequate distribution schemes: *user-perceives latency* (human interaction time) and *energy consumption on mobile devices*. Both profoundly influence the usability and the efficiency of pervasive applications in general.

Human Interaction Time

The interaction between a human and a flow can adhere to different patterns. In the simplest of cases, a human is notified by the workflow system about currently available activities and provides some sort of feedback when he has completed them. However, more complex interaction patterns may require a human to query the workflow system for information throughout the execution of the activities. The time needed to provide the desired information is experienced by the human as *waiting time*. An important design principle from the area of Pervasive Computing is to support humans as unobtrusively as possible [**?**, 67]. Therefore, applying workflows in this area requires that such waiting times are minimized.

The time is dependent on two factors: First, on the runtime of services that need to be executed in order to provide a human with the desired information. Second, on the time required to transfer data between workflow servers and service hosts participating in the execution of a workflow. New technologies like Cloud Computing support organizations in focusing on their core business [7] and lead to the necessity of using remote service providers within workflows. However, communicating data to remote networks may create extensive waiting times due to limited bandwidth and significant propagation delay.

In order to overcome this issue, we present an algorithm for distributing a workflow over a set of workflow servers such that the interaction time experienced by humans is minimized. Existing approaches for workflow optimization do not take this factor into account [69, 13, 32]. Our algorithm is based on a two-phase list-scheduling approach. In phase 1, an initial distribution is computed that is based on estimated values for activity execution and data transfer times. In phase 2, the initial solution is refined based on a hill-climbing algorithm. We show that our algorithm improves the interaction time

between humans and workflows by up to 80% compared to an approach in which the complete workflow is run on a single machine. Furthermore, we report an improvement of up to 10% compared to an existing greedy approach. We also show that our algorithm scales better with an increasing number of tasks compared to this approach.

Energy-Consumption

Driven by the advances in the area of pervasive computing [**?**], humans are no longer tied to their desk. They can use mobile devices to interact with workflows virtually anywhere. While this enables unobtrusive interactions with the workflow system, mobile devices have a constrained battery lifetime such that the energy consumption is a critical point for their usability. High energy consumption causes batteries to be drained fast, disrupting the business process and leaving users unable to continue their work. This renders the process less efficient and, thus, incurs higher costs. Alternatively, devices could be fitted with larger batteries to avoid such disruptions. However, this also incurs higher costs. Therefore, preserving energy is an important goal with respect to seamless workflow execution and cost reduction.

Today's workflow systems are usually deployed in a back-end infrastructure, such that the workflows need to communicate with mobile humans over a wireless medium. Due to this deployment scenario, extensive data transmission between user devices and workflows running in the infrastructure is required for each interaction with a mobile user. Consequently, this results in high energy costs since sending and receiving data are highly energy-intensive operations with current wireless communication technologies such as GPRS, UMTS or WiFi [10].

As a response to these challenges of mobile worfklow technology, we present an algorithm for distributing fragments of workflows from the infrastructure to mobile user devices in order to reduce these energy costs. Distributing workflows to mobile devices avoids transmissions of large volumes of data since this data is processed locally. Our algorithm constructs a *cost graph* that is based on the workflow model and takes two sources of energy into account: the data communication costs and the costs of service execution on the mobile device. We partition the cost graph using a minimum cut algorithm such that the workflow execution causes minimum energy consumption. Each partition of the cost graph contains the workflow activities to be executed either in the infrastructure or on the mobile devices. Our evaluations show that this approach achieves average energy savings of 37% for GPRS and 32% for UMTS communication compared to the centralized infrastructure-based approach. Existing work in the area of workflow distribution [13, 69] only focuses on infrastructure nodes, but does not consider the impact of workflow execution on the energy budget of mobile devices.

2.1 Workflow Model

A workflow is a directed acyclic graph $W = (A, \lambda, F, \rho, \theta_A, \theta_D)$. A denotes the set of activities in the workflow. The functionality of an activity is defined by means of the function $\lambda : A \rightarrow S$. This function binds an activity $a \in A$ to a required service $\lambda(a) \in S$, where S denotes the set of all services used by the workflow. The control flow of W is specified by means of the relation $F \subset A \times A$. The control flow defines the logical order of activities for the execution of the workflow. We refer to activities that model conditional or parallel behavior as *structural activities.* A conditional and parallel split is modeled as an activity with more than one outgoing control flow link. The set of outgoing control flow links of an activity $a \in A$ is denoted as F_a. For a given control flow link $f = (a_i, a_j)$, ρ_f is the probability that a_j will be executed after the completion of a_i. This value can be derived from execution traces of the respective workflow. For a conditional split, the workflow is executed following only a single alternative, i.e. the conditions $|F_a| > 1$ and $\sum_{f \in F_a} \rho(f) = 1.0$ hold. For a parallel split, all outgoing branches are executed in parallel, i.e. the conditions $|F_a| > 1$ and $\forall f \in F_a : \rho_f = 1$ hold. The latter also holds for all other links

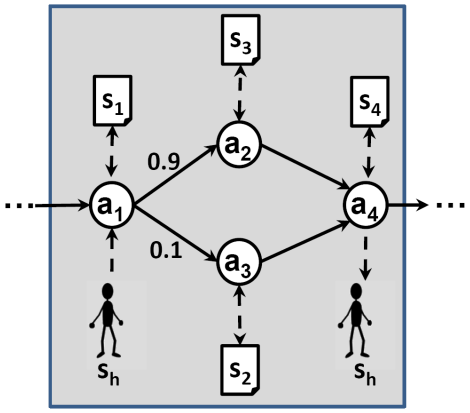


Figure 2.1: Human Interaction Pattern

originating from a non-structural activity.

The data flow of W is described by the two relations $L_{\mathrm{AA}} \subset A \times A$ and $L_{\mathrm{AS}} \subset A \times S$. Each $(a_i, a_j) \in L_{\mathrm{AA}}$ denotes a data link between two activities $a_i, a_j \in A$ and each $(a, s) \in L_{\mathrm{AS}}$ denotes a data link between an activity a and its associated service $s = \lambda(a)$. Over each of the data links a certain amount of data is communicated. The amount of the data that needs to be transferred for a service call $(a, s) \in L_{\mathrm{AS}}$ is denoted as $\theta_S(a, s)$. We assume that $\theta_S(a, s)$ covers the input as well as the output of the respective service call. Similarly, $\theta_A(a_i, a_j)$ specifies the amount of data that has to be exchanged between two activities $(a_i, a_j) \in L_{\mathrm{AA}}$. We assume that both θ_A and θ_S are available to our workflow distribution algorithm and can be either obtained by a workflow designer being an expert in the application domain or by analysis from histories of past execution traces of workflow instances.

A *Human Interaction Pattern* (HIP) is a connected subgraph of a workflow. It starts with a single entry activity which expects input data from a human and ends with a single exit activity which generates output data for the same human. This is a natural assumption as we focus on interaction patterns that resemble queries. An example for a HIP is given in Figure 2.1. Arrows show control flow links while circles and rectangles represent activities and services, respectively. The single entry activity is a_1 (a conditional split). The single exit activity is a_4. Note that there may be several HIPs in a single workflow.

2.2 Related Work

The task of workflow distribution is a hot topic of research within the workflow community. Each of the existing approaches for decentralized workflow management is heavily influenced by the class of performance gain to be achieved. In the following, we will compare our workflow distribution algorithms with existing approaches from related work and point out our unique contributions. First, we concentrate on algorithms that distribute workflows within a distributed network of wired workflow servers. Then, we discuss approaches with focus on mobile workflows, where flow-based applications run on mobile devices carried by human users.

Bauer and Dadam [13] propose an algorithm to reduce the network load produced by the workflow system. For this purpose, they have defined a probabilistic model that reflects the communication costs for workflow execution. By means of variable server assignments they create relations between activities which are used to select a suitable server in the infrastructure and distribute the workflow. First, each activity is greedily placed on a workflow server that minimizes the cost for its execution. Then, a hill-climbing algorithm is used to eliminate data transfers between neighbouring activities which have been placed on different servers. However, the initial distribution is calculated on a simple heuristics that considers flow activities isolated from each other. In contrast, we make use of the data flow dependencies

for the calculation of the initial placement. Due to his approach, we achieve better results for the network delay resulting from workflow execution as our evaluation shows. Son et al. [69] propose an algorithm for minimizing communication cost based on multi-level graph partitioning. A workflow is divided into several fragments. However, their solution assumes homogeneous communication links which is not valid in the Internet. In parallel computing, tasks have to be assigned to CPUs in order to optimize their execution. Many solutions assume that activities are not depending on each other, which leads to a Bin-Packing problem. Obviously, this assumption does not hold for our problem. More sophisticated approaches apply list scheduling algorithms [45, 15]. We adopted the basic idea of these algorithms while dropping their basic assumption of homogeneous communication links. In the area of grid computing, list scheduling algorithms are employed under the assumption of heterogeneous network links among loosely coupled computing systems [32, 63]. However, these algorithms assume variable task execution times to have a major influence on the overall execution of the task graph. In our scenario, we assume that tasks are services and that quality of service guarantees specify the time required for their execution. Hence, in our case the overall execution time mainly depends on the communication links between workflow servers, rendering respective algorithms like e.g. HEFT inappropriate.

For executing mobile workflows, Baresi et al. [11] propose graph partitioning rules to transform a centralized workflow model into a set of distributed fragments that run on different devices. The partitioning rules operate on an abstract graph representation of BPEL [40] to create a set of cooperating workflows. However, while this enables the creation of valid workflow fragments, the selection of these fragment has to be done manually by a workflow designer. Our approach supports a workflow designer to find the best fragments to be executed on mobile devices in terms of energy consumption. The results of our distribution algorithm can be used to apply the rules of Baresi et al. for cutting the workflow into fragments. Therefore, both approaches complement each other for creating a decentralized energy-efficient workflow executuion environment. MAUI [24] is a system for offloading code from a mobile device to an infrastructure in order to save energy. This is sensible if the energy required for executing the code exceeds the energy needed to transfer its state. However, MAUI works on a fine-grained level of individual functions. A workflow is a coarse-grained orchestration of large pieces of code (services). Thus, MAUI may be used orthogonally to our solution on the level of individual services. AIDE [54] is a distributed platform to offload application code from mobile devices to nearby powerful computers. For this purpose, the execution history of the application is monitored to create a fully connected graph of the application's interaction behaviour. The graph is then partitioned using a modified version of a minimum-cut algorithm, which considers the resource limitations on the mobile device such as the memory available for executing the application. For making distribution decisions it is assumed that applications can be freely migrated among different devices. In contrast, workflows often depend on services that are bound to a specific device, while other services are generally available on all devices. We explicitly deal with these restrictions in our approach for the distribution of a workflow. Furthermore, we have devised an approach for workflow distribution among an arbitrary number of devices, while AIDE can only divide an application into exactly two partitions. We conclude that workflow distribution in a heterogeneous environment for mobile and infrastructure-based devices is of major importance to conserve scarce resources on mobile devices. Existing approaches deal with workflow distribution and energy conservation under different assumptions and, thus, are not applicable to our problem.

2.3 Minimizing Human Interaction Time

2.3.1 Network Model

We assume a set of domains D, each representing a local network consisting of a set of hosts. A domain $d \in D$ provides a set of *service types* $\mathcal{S}_d$. Services of the same type may be available (replicated) in different domains and $\mathcal{S} = \bigcup_{\forall d \in D} \mathcal{S}_d$. Note that we model the functionality a human h provides as a special service $s_h \in \mathcal{S}$. We assume a workflow server and a *domain controller* in each domain.

The domain controller serves as an information service. It provides the link properties of all relevant communication links and a service discovery mechanism. This can be achieved by means of an overlay network between all domain controllers in the network. Services, workflow server and domain controller may be replicated inside a domain to allow for provisioning of quality of service guarantees, but for simplicity we treat each of them as being unique within the respective domain.

We assume that each domain is able to communicate to any other domain via the Internet. We denote the bandwidth and propagation delay between two arbitrary domains $d_1, d_2 \in D$ as $\beta(d_1, d_2)$ and $\delta(d_1, d_2)$, respectively. The bandwidth and propagation-delay between two hosts in the same domain is assumed to be constant and denoted as $\beta(d, d)$ and $\delta(d, d)$, respectively. We assume that $\forall d \in D$, $\forall d_i, d_j \in D$, $d_i \neq d_j : \beta(d, d) \gg \beta(d_i, d_j) \wedge \delta(d, d) \ll \delta(d_i, d_j)$, i.e. inter-domain delay dominates intra-domain-delay which reflects typical communication properties found in interconnected LANs.

2.3.2 Problem description

Our goal is to find a mapping function $\mu : A \rightarrow D$ of activities to domains that minimizes the average required communication time for all HIPs in a workflow. We focus only on activities that are part of a HIP. All other activities may be distributed according to other optimization goals (e.g. network load). Each execution of the workflow takes only a single *route* through the workflow. A route is a connected subgraph of a workflow that contains all activities visited in one execution. For example, a_1, a_2, a_4 as well as a_1, a_3, a_4 are both valid routes in the HIP shown in Figure 2.1.

Because we do not know this route in advance, we cannot optimize our mappings for it. Therefore, we solve the most general case and aim for minimizing the expected execution time of a HIP. Let R be the set of all routes of a HIP. The probability for the execution of $r \in R$ can be calculated as $\varrho_r = \prod_{\forall c \in r} \rho_f$. Furthermore, let φ_r^μ be a function that defines the execution time for r under the mapping μ. Then, our goal is to find a mapping μ such that the following sum is minimized:

$$\sum_{\forall r \in R} \varrho_r \cdot \varphi_r^\mu \tag{2.1}$$

In the following, we describe how φ_r^μ is calculated. In a parallel split, only the branch which results in the largest execution time is relevant for the overall execution time of the flow. We refer to the subgraph of r that contains only this longest branch for every parallel split as the *critical path* of a route. The time to execute a route of a HIP is influenced by three factors: The time κ_A required to transfer data between the activities, the time κ_S required to transfer data between activities and their mapped services, and by the time κ_X required to execute the corresponding services. Thus, we have $\varphi_r^\mu = \kappa_A + \kappa_S + \kappa_X$.

For the computation of κ_A, we have to distinguish two cases. First, two activities that exchange data may be mapped to the same domain and, thus, to the same workflow server according to our system model. In this case, κ_A is negligible because no data has to be sent over the network. Second, two activities may be mapped to different domains. In this case, the time required equals the sum of the propagation delay of the communication link between the respective domains and the time required for transmitting the data on the respective data flow link. Let $L_{crit} \subseteq L_{AA} \cup L_{AS}$ be the set of data links on the critical path of a route r, then

$$\kappa_A = \sum_{\forall (a_i, a_j) \in L_{crit}} \delta(\mu(a_i), \mu(a_j)) + \frac{\theta_A(a_i, a_j)}{\beta(\mu(a_i), \mu(a_j))}. \tag{2.2}$$

For the computation of κ_S, we proceed analogously. Given an activity a placed in domain d, let $\xi(a)$ be a function that returns the domain, among all domains that provide an instance of service type $\lambda(a)$, which can be accessed with lowest interaction time (ideally, $\mu(a) = \xi(a)$). Then,

$$\kappa_S = \sum_{\forall (a,s) \in L_{\text{crit}}} \delta(\mu(a), \xi(a)) + \frac{\theta_S(a, s)}{\beta(\mu(a), \xi(a))}. \tag{2.3}$$

We assume that service providers guarantee a certain execution time as part of a SLA. For the computation of κ_X, we accumulate the expected runtime required for the services mapped to the activities on the critical path.

Our problem is a generalization of the problem of task allocation in heterogeneous distributed systems (TAHDS) which is known to be NP-hard [45]. In TAHDS, we have a set of processors P and a set of tasks T. Two arbitrary tasks $i, j \in T$ have communication costs c_{ij}, and e_{ip} represents the cost of executing task i on processor p. The problem is to find a mapping of tasks to processors such that the sum of communication and execution costs is minimized. Note that communication costs between two tasks only occur if they are placed on different processors.

In the following, we reduce TAHDS to our problem in order to show that our problem is NP-hard as well. We map T to A and P to D, i.e. each task corresponds to an activity and each processor to a domain. We define a unique service for each activity and replicate it on every domain. We set the propagation delay between and within domains to zero. Furthermore, we set the bandwidth within each domain to ∞, i.e. hosts within a domain can communicate instantly. The bandwidth between domains is set to a constant β_{const}. The time to execute service replica $s = \lambda(a)$ running on domain d is set to e_{ip} where a is the activity corresponding to task i and d the domain corresponding to processor p. Θ_A is chosen such that $\Theta_A(a_i, a_j)/\beta_{const} = c_{ij}$ where tasks i and j correspond to activities a_i and a_j, respectively. Obviously, an algorithm that is capable to solve our problem is also able to solve TAHDS and hence, our problem is NP-hard. Therefore, we propose to use a heuristic algorithm to solve the problem because an extensive search of an optimal placement for example by means of backtracking is not feasible.

2.3.3 Heuristic Placement Algorithm

We propose a 2-phase algorithm based on a list-scheduling approach in order to find a mapping μ that minimizes the runtime of HIPs. Since HIPs are independent of each other, we map each HIP separately.

As soon as activity a is mapped to domain d, activities with a communication dependency to a have to communicate with d. Thus, they should not be assigned to the best domains in a greedy fashion. Instead, we have to take this dependency into account and map each activity depending on its influence on the runtime of a HIP: The more influence it has on the runtime, the earlier it is mapped.

According to our system model, the bandwidth and propagation delay between domains vary and the time required for communication depends on the network link used. Additionally, there may be services which are available in many domains while other services are only available in few domains. Therefore, it is not known which data link is the most expensive (in terms of communication time) until all activities have been mapped. Therefore, we use a heuristic to estimate the cost of each data link before the actual mapping. Since HIPs are independent of each other, we run the algorithm for each HIP in a workflow as soon as it is known from which domain the human accesses the workflow.

In a first phase, we assign weights to the data links to reflect their estimated costs. Then, we sort the links in descending order of their weights to ensure expensive activities are mapped to domains first. To derive an initial mapping, we iterate over the sorted list and map the activities to domains such that their execution time is minimized. The final mapping is created by optimizing the initial mapping through hill-climbing. The overall algorithm (called *Link Weight Activity Assignment* (LWAA)) is depicted in Listing 1.

2.3.3.1 Weighting and Ordering

The weight of a data link l for the initial mapping is calculated by virtually placing l on each of the possible network links between any pair of domains and by calculating the average time consumed over all these virtual mappings.

Let $l_{AA} = (a_i, a_j)$ be a data link between two activities and let $l_{AS} = (a, \lambda(a))$ be a data link between an activity and its required service. We distinguish between the average time $weight_W(l_{AA})$ required to communicate between workflow servers and the average time $weight_S(l_{AS})$ required to access a service

Listing 1 LWAA Algorithm

```
1:  // Let μ̂ be the associative array that represents μ
2:  // At the beginning ∀a : μ̂(a) =⊥ holds
3:  L_DF = weightDataLinks(L_AA ∪ L_AS)
4:  /* Initial mapping */
5:  while L_DF ≠ {} do
6:     l = Link with highest weight in L_DF
7:     if l ∈ L_AA then
8:        handleActivityToActivityDataLink(l, L_DF, μ̂)
9:     else if l ∈ L_AS then
10:       handleActivityToServiceDataLink(l, L_DF, μ̂)
11:    end if
12:    L_DF = L_DF \ {l}
13: end while
14: /* Optimize mapping */
15: hillClimbing()
```

from a workflow server that controls the corresponding activity. To compute $weight_W(l_{AA})$, we consider every possible mapping of two activities a_i and a_j:

$$weight_W(l_{AA}) = \frac{1}{|D|^2} \sum_{\forall d_k, d_l \in D: k \neq l} \frac{\theta_A(a_i, a_j)}{\beta(d_k, d_l)} + \delta(d_k, d_l) \tag{2.4}$$

Note that if both activities are mapped to the same domain, no data needs to be transferred.

Similarly, we compute an estimate of the delay created by service data links. In this case, we consider all possible mappings and calculate the average transmission time:

$$weight_S(l_{AS}) = \frac{1}{|D|} \sum_{\forall d \in D} \frac{\theta_S(a, s)}{\beta(d, \xi(a))} + \delta(d, \xi(a)) \tag{2.5}$$

The resulting list of data links is sorted in descending order.

2.3.3.2 Initial mapping

For the initial mapping of each activity to a domain, the algorithm proceeds through the list of data links, in descending order of their weights and maps each activity that has not already been processed. Links connecting two activities (L_{AA}) and links connecting an activity to a service (L_{AS}) are handled differently (cf. Listing 1 lines 8 and 10).

Listing 2 shows the handler procedure for links $(a, s) \in L_{AS}$. This handler finds the domain d that exhibits the least cost for calling service s residing in d when placing activity a in d. Listing 3 shows how to handle a data link $(a, a) \in L_{AA}$. We aim at placing both activities in the same domain such that no data has to be transferred over the network. However, we also do not want to reduce the degree of freedom for the placement more than required. We have to distinguish three different cases: 1. None of the activities is mapped 2. Only one of the activities is mapped 3. Both activities are mapped.

Listing 2 Handle activity to service data link

procedure *handleActivityToServiceDataLink*$(l, L_{DF}, \hat{\mu})$

```
1: (a, s) := l //get corresponding service and activity
2: μ̂(a) := MinArg_{d∈D:s∈S_d} δ(d, d) + θ_S(a,s) / β(d,d))
```

Listing 3 Handle activity to activity data link

procedure *handleActivityToActivityDataLink*$(l, L_{DF}, \hat{\mu})$

1: $(a_i, a_j) := l$ //get activities the data link connects
2: **if** $(\hat{\mu}(a_i) = \perp) \wedge (\hat{\mu}(a_j) = \perp)$ // Both activities unmapped **then**
3: **if** $\exists d \in D : \lambda(a_i) \in \mathcal{S}_d \wedge \lambda(a_j) \in \mathcal{S}_d$ **then**
4: $a' := Merge(a_i, a_j)$
5: // Sorted insert of new service data link
6: $L_{\text{DF}} := L_{\text{DF}} \cup \{(a', \lambda(a'))\}$
7: // remove service links of a_i and a_j
8: $L_{\text{DF}} := L_{\text{DF}} \setminus \{(a_i, \lambda(a_i)), (a_j, \lambda(a_j))\}$
9: **end if**
10: **else if** $(\hat{\mu}(a_i) \neq \perp) \wedge (\hat{\mu}(a_j) = \perp)$ //First activity mapped **then**
11: **if** $\lambda(a_j) \in \mathcal{S}_{\hat{\mu}(a_i)}$ **then**
12: $\hat{\mu}(a_j) := \hat{\mu}(a_i)$
13: **end if**
14: **else if** $\hat{\mu}(a_i) = \perp) \wedge (\hat{\mu}(a_j) \neq \perp)$ //Second act. mapped **then**
15: **if** $\lambda(a_i) \in \mathcal{S}_{\hat{\mu}(a_j)}$ **then**
16: $\hat{\mu}(a_i) := \hat{\mu}(a_j)$
17: **end if**
18: **end if**
19: // If both activities are mapped nothing has to be done.

The first case is handled in lines 2 – 9 of Listing 3: we check if there exists a domain hosting both service types required by the activities. If such a domain exists, we merge both activities.

The procedure of merging is depicted in Figure 2.2. The result of merging two activities $a_i, a_j \in A$ is a new activity a' with a corresponding data link to a virtual service $\lambda(a') = s'$ which serves as a container for both $\lambda(a_i)$ and $\lambda(a_j)$. Each operation performed on s' has to be performed for all services in s'. This is illustrated in Figure 2.2. a_2 and a_3 are merged into a new activity a' with a data link to service type $s' = \{s_2, s_3\}$. The weight of the newly created data link is the sum of the weights of the original links. All other links remain unchanged. Note, that we do not map the merged activities to a domain right away as it may be merged with further activities.

We only merge if there exists a domain hosting the services required by *both* activities because the service access of a_2 and a_3 needs to be restricted to their own domain in order to save communication time. Using the workflow in Figure 2.2 (left), we explain the rationale behind this idea. Assume that $D = \{d_1, d_2\}$ with $S_{d_1} = \{s_1, s_2\}$ and $S_{d_2} = \{s_3, s_4\}$ and none of the activities is currently assigned to any

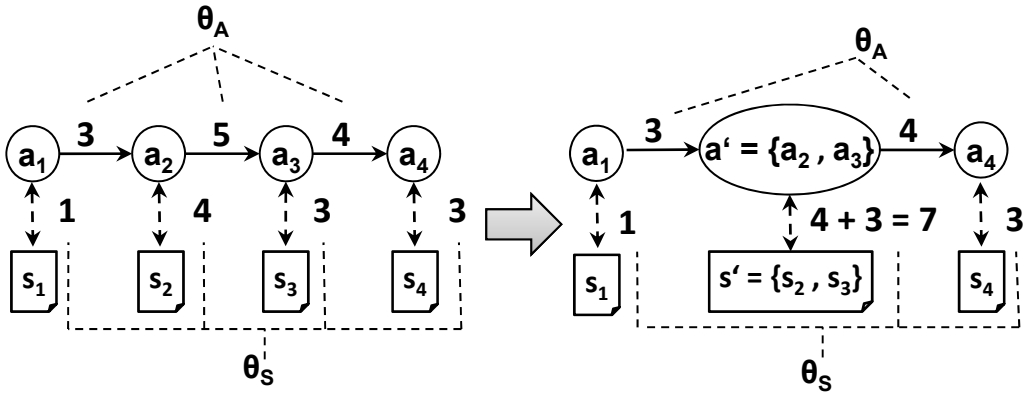


Figure 2.2: Merging of unassigned activities

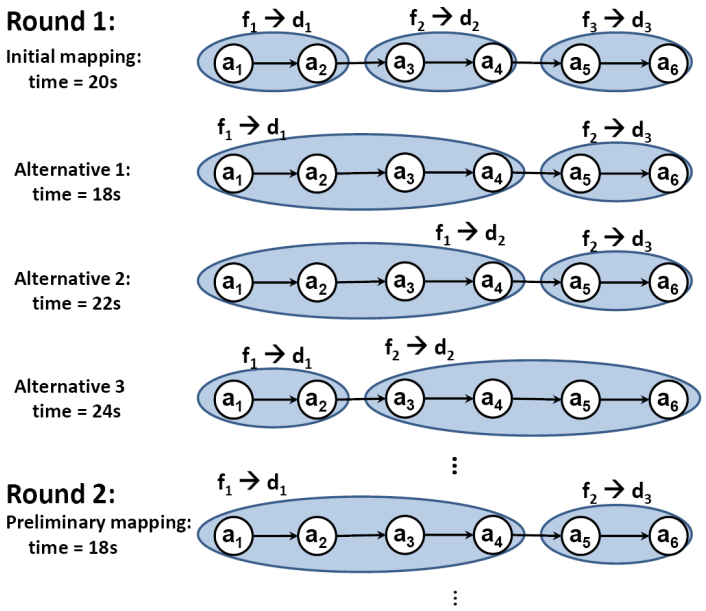


Figure 2.3: Hill-climbing

domain. According to the ranking of data links, the link between (a_2, a_3) has to be processed first. If created a merger $a' = \{a_2, a_3\}$, we would have to map a' either to d_1 or d_2. Thus, either (a_2, s_2) or (a_3, s_3) would be mapped to an inter-domain link because there exists no domain hosting s_2 *and* s_3. The data transferred via the link (a_2, a_3) is either the output data of s_2 or the input data for s_3. Consequently, we would omit the time required to transfer the data between both activities. However, we would have to transmit the same amount of data via a communication link in the global network which would require the same amount of time that has been saved. Furthermore, through merging in this case we would limit the degree of freedom as, afterwards, we could not map a_2 and a_3 separately.

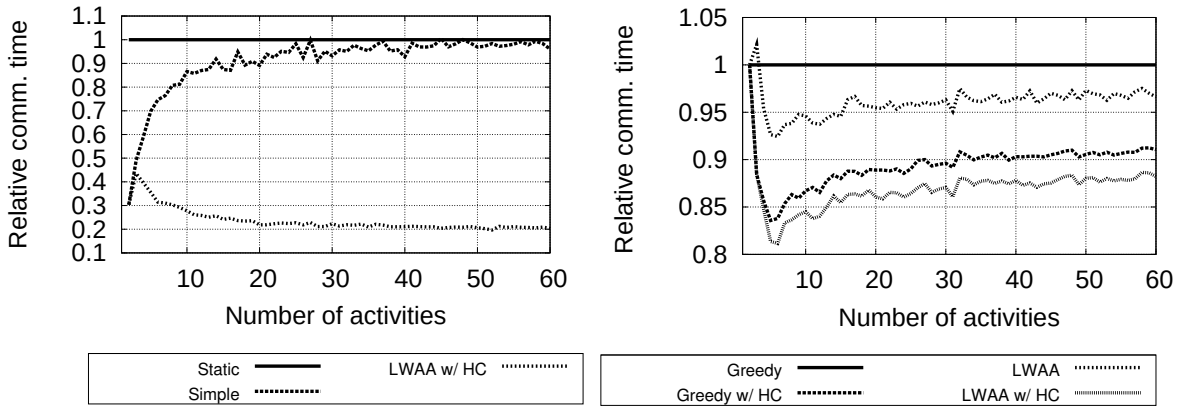
It may happen that only one of both activities is already mapped to a domain. This is covered in lines 10 to 18 of Listing 3. Analogously to the previous case, we map the unmapped activity to the domain of the already mapped activity only if this domain hosts the required service. Finally, it is also possible that both activities are already mapped to a domain. In this case, we do nothing since the currently handled data link must have a lower priority than the data links that led to a mapping of the respective activities to domains.

2.3.3.3 Optimized mapping

After the initial mapping is completed, we adjust it to the actual bandwidth/propagation delay in the network using a hill climbing algorithm in oder to further reduce the time consumed by transferring data via the global network. The principle of the algorithm is depicted in Figure 2.3.

First, we extract the clusters of the initial mapping. A cluster $A_F \subset A$ is the largest set of activities that form a connected graph with $\forall a_i, a_j \in A_F : \mu(a_i) = \mu(a_j)$. In Figure 2.3, there exist three clusters in the initial mapping, namely f_1, f_2 and f_3. We calculate the time required for the HIP if all activities of a cluster are mapped first to the domain of its preceding and then to the domain of its succeeding cluster. The possible alternatives and the time required for each alternative are depicted in rows 2 to 4 of Figure 2.3. The best alternative is selected as new preliminary mapping. This procedure is repeated until no mapping which requires less time can be found.

We only remap complete clusters because it is unlikely that remapping single activities results in a performance gain. If this would be the case the initial mapping would have come to a different conclusion.



(a) Non-partitioned placement (b) Partitioned placement

Figure 2.4: Comparison with other placement approaches

2.3.4 Evaluation

In this section, we describe our evaluation setup and results. We used a simulation approach and modelled synthetic networks as well as workflows that follow our models discussed in Sections 2.1 and 2.3.1. As the algorithm is executed for each HIP separately, we restrict the generation process to a workflow consisting of a single HIP. The number of activities between the human activities varies between 0 and 60.

We simulate 50 different domains. The bandwidth for communication within each domain is set to 1 GBit/s assuming a Gigabit Ethernet. For the communication between humans and workflow servers, we assume a 54 MBit/s WLAN connection. The bandwidth of communication links between domains is set to be between 2 MBit/s (E1) and 34 MBit/s (E3) to reflect the SLAs between service providers.

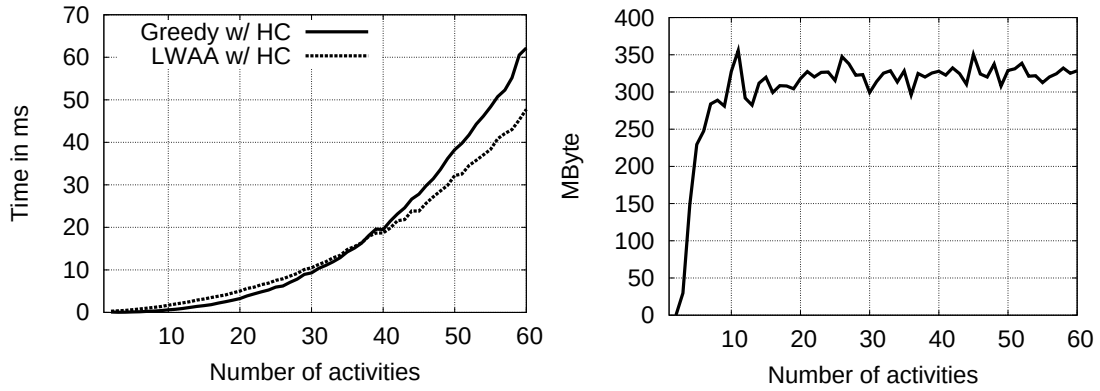
We assumed a uniform delay to simplify our simulation. Since the delay between domains is only influencing the ordering of the weighted data links, this does not change the qualitative results.

We have 200 service replicas drawn from 20 services according to a Zipf distribution. The services are randomly assigned to the 50 domains. We use a Zipf distribution because there may be few very popular services available in many domains, while there are many more specialized services which are only provided in a few domains.

For the generation of workflows we use a grammar that is able to generate sequences, conditional and parallel structures. The rules of the grammar are chosen randomly until the desired number of activities is reached. The values for θ_S are generated randomly according to a uniform distribution with a maximum of 100 MByte to allow for a wide variety of data flow links. The values for θ_A are implicitly defined by θ_S to guarantee a consistent data flow meaning that all data received by an activity is sent via its outgoing data flow links to other activities. The assignment of activities to (human) services is also chosen uniform randomly.

We compare our algorithm with four other approaches:

- *Static* maps all all activities of a HIP to a random domain.
- *Simple* maps all activities of a HIP to the human's current domain.
- *Greedy* proceeds through all activities, searches for the domain reachable with the highest bandwidth hosting the next required service and maps the activity to this domain.
- *Greedy w/ HC* enhance *Greedy* with a subsequent hill-climbing (cf. Figure 2.3). This is an adapted version of the algorithm proposed by Bauer et al. [13].



(a) Time consumption

(b) Tolerable data amount per migr.

Figure 2.5: Performance analysis

In Figure 2.4(a), we compare our distribution algorithm (*LWAA*) with the *non-partitioned approaches* (*Static* and *Simple*). We compared the relative gain of using our algorithm. The reference (at 1.0) is the *Static*. Figure 2.4(a) shows that, for an increasing number of activities per HIP, our algorithm quickly converges to around 20% of the time required for *Static*. This is because in the non-partitioned approaches a lot of expensive service calls have to use low quality network links. Thus, they consume a lot of time for communication as only the services that are located in the same domain can be accessed in a performant way.

Figure 2.4(b) depicts the effectiveness of our algorithm compared to the greedy approaches. The initial placement computed by our algorithm is between 12% and 16% better than the placement computed by the *Greedy* approach. This is due to the fact that our algorithm takes the data flow between activities into account and, thus, computes suitable clusters which is not done in the *Greedy* approach. The *Greedy w/ HC* approach performs better than the *LWAA* algorithm without subsequent hill-climbing. This is due to the fact that clusters are assigned to domains without taking the communication links of the individual domain into account. The results show that our algorithm is better compared to the *Greedy w/ HC* approach by 8% to 10% due to the better initial placement.

We compare our algorithm to the greedy approach in terms of the required computation time in Figure 2.5(a). Both algorithms show very similar execution times at first, until the effort for the subsequent hill-climbing starts to dominate at around 35 activities. This is because of the fact that *LWAA w/ HC* builds activity clusters, reducing the number of clusters left for the hill-climbing compared to *Greedy w/ HC*.

If the source activity of a data link is mapped to another domain than its target activity, data has to be transferred between workflow servers during the execution of a workflow. This process is called *migration*. The amount of data that has to be transferred in a migration may differ, for example, depending on whether the workflow management system has to transfer additional logging data for compensations. This is not accounted for in our simulation and would impact the performance of our algorithm negatively. Therefore, we measured the amount of data that can be sent additionally for each migration before the *Static* or *Simple* approach outperform our distribution approach. Figure 2.5(b) shows that this amount can be about three times the maximum amount of data that occurs in the data flow, indicating that considerable migration overhead can be tolerated by our algorithm.

2.4 Minimizing Energy Consumption

2.4.1 Application Scenario

The application scenario depicted in Figure 2.6 models a quality assurance workflow for car shipment management (logistics domain). In this scenario, we consider a big harbor where ships deliver large

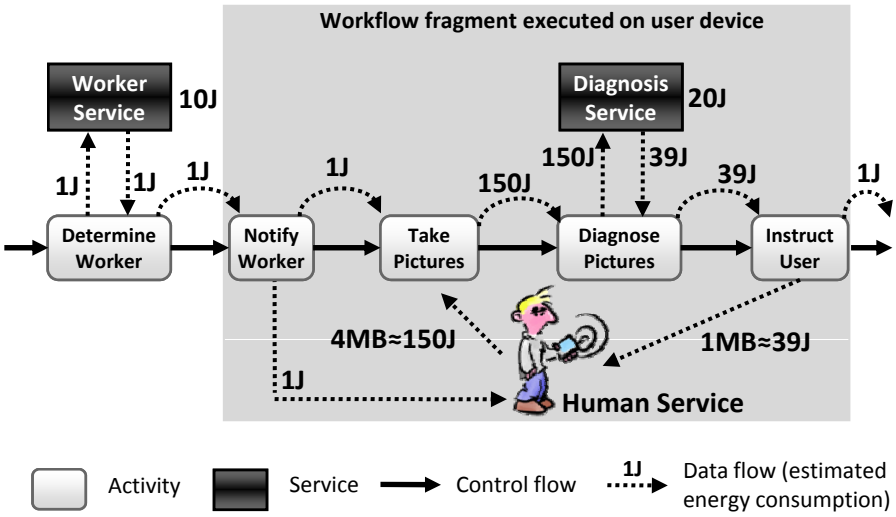


Figure 2.6: Example workflow with annotated energy costs

numbers of cars. The cars go through a series of treatments that involve e.g. fitting radios, maintenance, and cleaning activities. Cars may be parked for certain periods of time and retrieved for additional treatments or onward shipment. During this whole process, cars may be damaged. Therefore, the personnel that is handling the cars (handling drivers) needs to check them regularly for damages. Based on the results of these checks, cars may be sent back to the manufacturer, they may be sent to a repair station, or insurance cases may have to be filed. Since the handling drivers are no experts in detecting and evaluating damages, this is done by taking photos of the car and sending these photos to some image recognition service that is trained to detect specific kinds of damages.

First, an available worker is chosen by the workflow from a database using the *Worker Service*. The chosen worker gets instructions (*Notify Worker*) and interacts with the workflow by taking pictures of a car and submitting them to a diagnosis service. If the service detects a problem with the car, the worker is instructed to take further steps to fix the problem. For example, this can be an instruction manual or video.

In our example, the mobile device has to transmit 4 MB of data and receives 1 MB. Assuming that the workflow is executed in the infrastructure, this drains about 140J via UMTS and even 189J (150J+39J) via GPRS (shown in the figure) [10] of energy from the mobile device. We can save a significant amount of energy by executing the workflow (including the diagnosis service) on the users' mobile device, since data transmission dominates service execution in terms of energy usage. For the implementation of the diagnosis service on a mobile device, several approaches such as Neuronal Networks or Markov models are feasible. These models can be trained online and enable real-time recognition on mobile devices [18]. Executing the shaded area of the workflow on the mobile device costs only 22J (20J for executing the diagnosis service on the mobile device and 2J for small data transmissions) as opposed to the 189J consumed with an infrastructure-based service. We save 167J. Saving this amount of energy for every car the worker handles has a strong impact on the battery lifetime of his device.

Our assumption is that off-the-shelf smartphone technology will be used for applications like the one investigated here. The performance and the form factor of these devices is adequate. Using off-the-shelf phones has several advantages:

- They are generally cheaper than special-purpose devices that have been designed for a very small market segment.

- They are constantly improved, removing the necessity to explicitly invest in costly redesigns.
- They offer a very flexible hardware platform with different communication devices and sensors.
- They offer additional communication functionality and applications that can support the general work process and enable integration.
- They are available in arbitrary numbers such that scaling the system up is not a problem in terms of the hardware.

Of course, precautions in terms of security have to be taken. But these can be implemented in software.

Typical smartphones have a battery capacity of about 17000J and the typical energy consumption of smartphone users is between 144J and 3600J per hour with a median of about 850J [26]. Hence, centralized workflow execution would drain the overall energy of a mobile device heavily in our scenario.

2.4.2 Network Model

The network consists of a set of mobile devices $D = \{d_1, d_2, \ldots\}$, where each device corresponds to a human user who participates in the execution of the workflow. Furthermore, the network consist of an abstraction of the back-end infrastructure inf. We abstract from the concrete implementation of this infrastructure since it does not influence the energy consumption of mobile devices. For example, the infrastructure may be a standalone server, a server cluster or even a distributed network of servers. The set of all hosts is denoted as $H = D \cup \{inf\}$. Thus, the term *host* may refer to both, infrastructure or mobile device. We assume a communication system between hosts, e.g. cellular communication (UMTS, GPRS) or WiFi via access points, such that each pair of hosts can communicate.

The set of services S accessible on the network hosts consists of 3 disjoint subsets S_{inf}, S_{human} and S_{mov} that represent different service classes. Each service $s \in S_{inf}$ is a computational task executed by piece of software, which is only available in the infrastructure, e.g., due to a large database which needs to be accessed. These kind of services are called *infrastructure* services. In contrast to this, each service $s \in S_{human}$ represents a task that is to be performed by a human user (e.g., repairing a car). We refer to these services as *human service*. A human service corresponds to a mobile device which is used by the human to retrieve and send the information relevant for his task. Each services $s \in S_{mov}$ can be either executed in the infrastructure or on the mobile device. An example of such a *movable service* is the diagnosis service from our application scenario in Section 2.4.1. The decision where to execute a movable service (on the mobile device or on the infrastructure) depends on the outcome of our workflow distribution algorithm.

2.4.3 Problem Description

Our goal is to find a distribution of a workflow that minimizes the energy consumption for the workflow execution on the mobile devices. More formally, a possible workflow distribution can be described by a function $\mu_1 : A \rightarrow H$ that maps each activity in the workflow to a host that shall execute it. At the same time, we have to decide for the host where to execute a movable service. Formally, this is expressed by a second mapping function $\mu_2 : S_{mov} \rightarrow H$. Among all possible mappings, we are interested in the most energy-efficient mappings μ_1^* and μ_2^*, i.e., the mappings that minimize the sum of the drained energy of all mobile devices $d \in D$.

The total energy cost for workflow execution under the given mappings μ_1 and μ_2 is denoted as $E_{total}(\mu_1, \mu_2)$. The cost is the sum of the energy consumed for executing movable services on the mobile devices as well as the energy spent for wireless communication. In the following, we refer to the energy

required to transmit k bytes as $E_T(k)$. In our evaluation, we will use existing energy models to compute $E_T(k)$ for specific wireless communication technologies, e.g., GPRS. The energy required for executing a movable service s is given by $E_X(s)$. In the following, we describe the cost model to determine the total energy cost for a workflow execution.

First, let us consider the energy cost $E'_X(s)$ for executing a movable service $s \in S_{mov}$. $E'_X(s)$ is defined as:

$$E'_X(s) = \begin{cases} E_X(s), & \text{if } \mu_2(s) \in D \\ 0, & \text{otherwise} \end{cases}$$

Each movable service $s \in S_{mov}$ may be executed on a mobile device or in the infrastructure. If s resides on a mobile device, energy has to be spent for its execution. Otherwise, if s is executed in the infrastructure, no energy is consumed on the mobile devices.

Second, we have to consider the energy costs for remote service calls. For each data link $(a, s) \in L_{AS}$ the consumed energy $E'_T(a, s)$ can be calculated as follows:

$$E'_T(a,s) = \begin{cases} E_T(\theta_S(a,s)), & \text{if } (\mu_1(a) \in D \wedge (s \in S_{inf} \vee \\ & s \in S_{mov} \wedge \mu_2(s) = inf) \vee \\ & \mu_1(a) = inf \wedge (s \in S_{human} \vee \\ & s \in S_{mov} \wedge \mu_2(s) \in D)) \\ 0, & \text{otherwise} \end{cases}$$

We have to spend energy costs, whenever the activity a resides on a host which is different from the host where its required service $\lambda(a)$ is executed. As a consequence, data needs to be transferred over the wireless medium. This may be true for different cases. If a workflow activity is assigned to a mobile device and the service can be found in the infrastructure (because it is an infrastructure service or a movable service running in the infrastructure), network communication is required. In the other case, whenever the activity is assigned to the infrastructure and the service resides on the mobile device (because it is an human service or a movable service executed on the mobile device), the data needs to be transferred over the network. We have no transmission costs only if the activity a and the service s are either running both on the same mobile device or both in the infrastructure.

Third, we have to consider the communication costs for the transmission of data among worfklow activities. For each such data link $(a_i, a_j) \in L_{\mathrm{AA}}$ the consumed energy $E'_T(a_i, a_j)$ is defined in the following manner:

$$E'_T(a_i, a_j) = \begin{cases} E_T(\theta_A(a_i, a_j)), & \text{if } \mu_1(a_i) \neq \mu_1(a_j) \\ 0, & \text{otherwise} \end{cases}$$

We only have to pay the energy costs in case the communicating activities do not reside on the same host. Then, the required data must be sent over the wireless medium from the preceding to the succeeding activity. In contrast to this, activities which are assigned to the same host do not produce any energy costs for communication.

The total energy required for the execution of a workflow under the given mappings μ_1 and μ_2 is then defined as:

$$E_{total}(\mu_1, \mu_2) = \sum_{\forall s \in S_{movable}} E'_X(s) + \sum_{\forall (a,s) \in L_{\mathrm{AS}}} E'_T(a, s) + \sum_{\forall (a_i, a_j) \in L_{\mathrm{AA}}} E'_T(a_i, a_j)$$

All sources of energy consumption are covered in this equation. In the following, we present our algorithm that minimizes this function by finding an optimal workflow distribution. In Section 2.4.4.3, we show that this algorithm minimizes both, the sum of energy consumption over all devices *and* the energy consumed individually on each device. Both optimization goals are equivalent in our case.

Algorithm 4 Cost Graph Construction

1: // Let $G = (V, E)$ be the cost graph to be constructed
2: $V := A \cup H$
3: **for all** $(a_i, a_j) \in L_{\mathrm{AA}}$ **do**
4: $E := E \cup \{(a_i, a_j, E_T(\Theta_A(a_i, a_j)))\}$
5: **end for**
6: **for all** $(a, s) \in L_{\mathrm{AS}}$ **do**
7: **if** $s \in S_{human}$ corresponding to device d_i **then**
8: $E := E \cup \{(d_i, a, E_T(\Theta_S(a, s)))\}$
9: **else if** $s \in S_{inf}$ **then**
10: $E := E \cup \{(a, inf, E_T(\Theta_S(a, s)))\}$
11: **else if** $E_T(\Theta_S(a, s)) > E_X(s)$ **then**
12: //service communication costs dominate - call service locally
13: $E := E \cup \{(a, inf, E_X(s))\}$
14: **else**
15: //service execution costs dominate - call service remotely
16: $E := E \cup \{(a, inf, E_T(\Theta_S(a, s)))\}$
17: **end if**
18: **end for**

2.4.4 Distribution Algorithm

Our approach for energy-efficient workflow distribution is divided into two steps. First, based on the network and workflow model we construct a *cost graph*, which models the energy costs for the workflow execution on the network hosts. The nodes of the cost graph are the activities in A and the hosts in H. Its edges are annotated with weights representing the energy costs resulting from data communication or from the execution of services on the mobile devices. Second, we use this cost graph as input to a minimum cut graph partitioning algorithm. The algorithm partitions the cost graph into $|H|$ subgraphs, where each subgraph contains exactly one host and zero or more activities. The set of activities contained in the subgraph represents the fragment of the workflow that is to be executed on the particular host contained in the subgraph. Since we apply a minimum cut approach to partition the workflow, we guarantee that the energy costs for the resulting placement are minimized. In the following, we describe each of the steps involved in more detail, and we prove the optimality of the approach.

2.4.4.1 Cost Graph Construction

The cost graph $G = (V, E)$ consists of a set of nodes V and a set of weighted edges E. The graph is constructed using Algorithm 4. Initially, we create a node in V for each host of the network and each activity of the workflow, i.e. $V = A \cup H$ (line 2). Then, the set of weighted edges E is determined, where an edge is created for each data link in the workflow (L_{AA} and L_{AS}). The weight w associated with an edge $(u, v, w) \in E$ represents the energy costs of a hypothetical placement, where the source node u and target node v of the edge are assigned to different partitions (which represent different hosts).

Weighted edges are created in the following manner. First, we handle the case of activity-to-activity communication and create an edge between any two different activities $a_i, a_j \in A$ that share a data dependency (lines 3-5). The edge (a_i,a_j) is weighted by the amount of energy required to transmit the data $\Theta_A(a_i, a_j)$ between the activities. Then, we distinguish between several cases for the invocation of service s by activity a. If the service is provided by a human, we introduce an edge weighted by the costs of transmitting the data $\Theta_S(a, s)$ between the activity and the mobile device corresponding to the human service s (lines 7-8). If the service runs in the infrastructure and is not available on the mobile

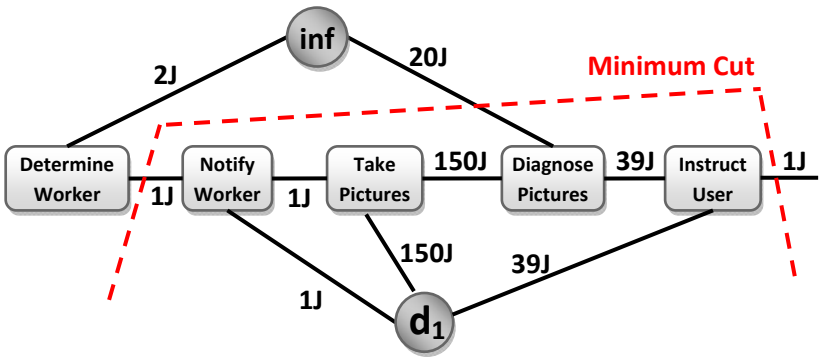


Figure 2.7: Cost graph created from the example workflow shown in Figure 2.6

device, we add an edge weighted with the costs of transmitting the data $\Theta_S(a, s)$ between the activity and the service s (lines 9-10). Finally, in case of a movable service, we have to compare the energy required to transmit the input/output data of the service to the infrastructure with the energy required for local service execution ($E_T(\Theta_S(a, s)) > E_X(s(a))$). The rationale is that a necessary criterion for running the service on the mobile device is that the service execution costs are lower than the communication costs. In case the service communication costs dominate the execution costs, the service s should always be executed where the corresponding activity a is placed (either infrastructure *or* mobile device). Hence, we must only consider the service execution costs if a is placed on a mobile device. Thus, we create an edge between a and the infrastructure node inf which is weighted by the service execution cost $E_X(s)$ (line 13). However, in case the service communication costs dominate, we create an edge between the activity and the infrastructure weighted by the communication costs (line 16).

As an example, Figure 2.7 shows the cost-graph constructed for the workflow in Figure 2.6. The nodes of the graph are the set of the activities, the infrastructure (denoted as inf), and the single device used by the human service (called d_1). Data flow links are simply mapped to edges in the graph with corresponding weights (Algorithm 4, lines 3-5), in the same way as the data flow links between activities and the user (Algorithm 4, lines 9-10). Since the execution of the 'worker service' on a mobile device causes more energy costs (10J) than the costs for the required data transmission (2J), an edge between the 'determine worker' activity and the infrastructure is created (Algorithm 4, lines 14-16). As the execution costs for the diagnosis service are smaller than the cost of transmitting its input/output, both the diagnosis service and the 'diagnose picture' activity should be executed at the same host. If the 'diagnose picture' activity is executed on the mobile device, we have to consider the execution costs of 20J. Hence, we have to create an edge with weight 20J (Algorithm 4, lines 11-13).

2.4.4.2 Workflow Distribution

For the purpose of workflow distribution, we compute a partitioning of the cost graph that assigns each activity to a host d and, thus, represents the desired mapping μ_1. Based on the activity mapping, we can also derive the assignment of movable services to hosts for the mapping μ_2.

We use the *minimum cut algorithm* [42, 20] to determine the partitioning that minimizes the consumed energy. A minimum cut creates exactly two partitions C and $V \setminus C$ of the graph. Partition C contains the host d and zero or more activities from the set $C \cap A$ assigned to it, and partition $V \setminus C$ contains all remaining hosts $H \setminus \{d\}$ and activities $(V \setminus C) \cap A$. The sum of the weights of all edges between different partitions represents the energy consumed by the mapping. Consequently, our goal is to determine the partitioning that minimizes the sum of the weights of all edges which are cut through the partitioning.

Algorithm 5 Workflow Distribution

1: // Let $G = (V, E)$ be the contructed cost graph
2: $V_{tmp} := V$
3: **for all** $d_i \in D$ **do**
4: $t' := merge(H \setminus \{d_i\})$
5: $(C, V \setminus C) = calculateMinimumCut(d_i, t')$
6: **for all** $a \in C \cap A$ **do**
7: $\mu_1(a) := d_i$
8: $V_{tmp} := V_{tmp} \setminus \{a\}$
9: **end for**
10: **end for**
11: // assign remaining activities to infrastructure
12: **for all** $a \in V_{tmp} \cap A$ **do**
13: $\mu_1(a) := inf$
14: **end for**
15: **for all** $(a, s) \in L_{\text{AS}}$ with $s \in S_{mov}$ **do**
16: **if** $\mu_1(a) \in D \wedge E_T(\Theta_S(a, s)) > E_X(s)$ **then**
17: $\mu_2(s) := \mu_1(a)$
18: **else**
19: $\mu_2(s) := inf$
20: **end if**
21: **end for**

The minimum cut is defined as follows: Given the cost graph $G = (V, E)$, the minimum $s - t$ cut of G is a partition $(C, V \setminus C)$ such that $s \in C$, $t \in V \setminus C$ and

$$\sum_{(u,v,w) \in E : u \in C \wedge v \in V \setminus C} w$$

is minimal among all possible partitions $C \subseteq V$. Several approaches have been proposed to find the minimum $s - t$ cut with polynomial time complexity [42, 20].

To solve our problem, we have to extend this algorithm to produce $|H|$ partitions instead of two. For this purpose, we propose Algorithm 5 that takes an iterative approach to compute the partition C with the activities for each mobile device d_i. In each iteration, we find what we call a minimum $s-T$ cut for mobile device d_i, i.e., a minimum cut that separates node d_i and all remaining hosts in the set $T = H \setminus \{d_i\}$. For this purpose, we modify the cost graph and merge all $t \in T$ into a new node t' such that all $(u, v, w) \in E$ with $v \in T$ are replaced by (u, t', w) (line 4). The idea is to create a new virtual node in the graph that represents all other hosts except for d_i. Thus, we can return to the two-partition cut problem and execute the minimum $s - t'$ cut to find the solution to our extended minimum $s - T$ cut problem (line 5). After this partitioning step, we place all activities which are part of d_i's partition on d_i (lines 6-9). We follow this approach for each mobile device. Afterwards, there may remain activities which have not been assigned to any mobile device. These activities are then assigned to the infrastructure (lines 12-14). Thus, we have created H partitions of the cost graph and we have found the desired mapping function μ_1 that creates an optimal mapping as we prove below. Based on this mapping, we can determine the placement of the movable services in S_{mov} on the hosts in H (lines 15-20). A movable service is placed on a mobile device only if its calling activity is also placed on the same device and the service execution costs are lower than the communication costs (lines 16-17). In all other cases, the service is executed in the infrastructure (line 19). Thus, we have found the required mapping for μ_2 and completed the workflow distribution.

For the cost graph in Figure 2.7, we execute the minimum-cut algorithm once, since only one mobile device is part of the scenario. The resulting cut is indicated as a dashed line. Three activities and the movable 'Diagnose Picture' service are executed on the mobile device d_1, resulting in $(1+20+1)J = 22J$ of consumed energy. Thus, we can achieve significant energy savings of $(1 + 39 + 150)J - 22J = 168J$ compared to the approach where the entire workflow is run in the infrastructure.

2.4.4.3 Optimality Discussion and Proof

In order to find the distribution of the workflow with minimal energy consumption, we calculate the minimum cut for each mobile device separately as explained in Section 2.4.4.2. Since each single cut is optimal, the overall system is also optimal if there are no conflicts, i.e. if there are *no overlapping cuts*. An overlap of two (or more) cuts (shaded area in Figure 2.8) means that at least one activity (residing in the overlapping region of both cuts) is claimed by two (d_i or d_j) or more devices to render the energy usage of each of those mobile devices minimal.

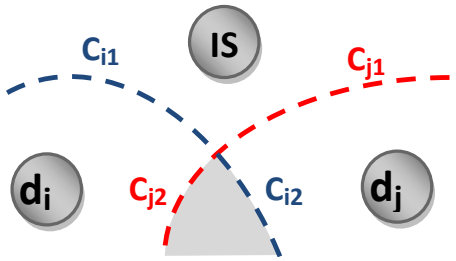


Figure 2.8: Two overlapping cuts

Of course, an optimal solution could still be found under these conditions simply by placing the activity that is requested by d_i and d_j on either one of the two such that it produces the lowest cost. However, if the overlap becomes complex, i.e. if more activities fall into overlap regions, this simple extension of the optimal cut algorithm would not suffice to create an optimal overall solution. Therefore, it suffices to show that our algorithm never produces overlapping cuts to prove its optimality.

We show that there is no activity that is member of two cuts, i.e. $\forall d_i, d_j \in D, d_i \neq d_j : C(d_i) \cap C_(d_j) = \emptyset$, where $C(d_i) = \{a \in A | \mu_1(a) = d_i\}$ is the set of activities which are placed on device $d_i \in D$ based on our algorithm.

Assume there are two overlapping cuts $C(d_i), C(d_j)$ as shown in Figure 2.8 with $C(d_i) \cap C(d_j) \neq \emptyset$. Let c_{i2} and c_{j2} denote the overlapping part and c_{i1} and c_{j1} the non-overlapping part of cuts $C(d_i)$ and $C(d_j)$, respectively. Let $w(c)$ be the sum of weights (i.e. the cost) of a cut c. Then, $w(c_{i1}) + w(c_{i2})$ and $w(c_{j1}) + w(c_{j2})$ are the costs of the minimum cuts for d_i and d_j respectively. Note that there must be at least one activity in the overlapping area (shaded).

Since the cut $c_{i1}c_{i2}$ was chosen as minimum cut between d_i and d_j/IS, we must have $w(c_{i1})+w(c_{i2}) \leq w(c_{i1}) + w(c_{j2})$, i.e. in particular $w(c_{i2}) \leq w(c_{j2})$. If $w(c_{i2}) = w(c_{j2})$, we can easily construct non-overlapping cuts for both d_i and d_j with minimum costs (actually $w(c_{i2}) = w(c_{j2})$ cannot happen if we assume non-zero execution costs for services, since the overlapping cut contains at least one activity). If $w(c_{i2}) < w(c_{j2})$, $c_{j1}c_{i2}$ would be a cut with less costs for d_j which contradicts the premise. Hence, there can be no overlapping cuts, which proves the optimality of our approach.

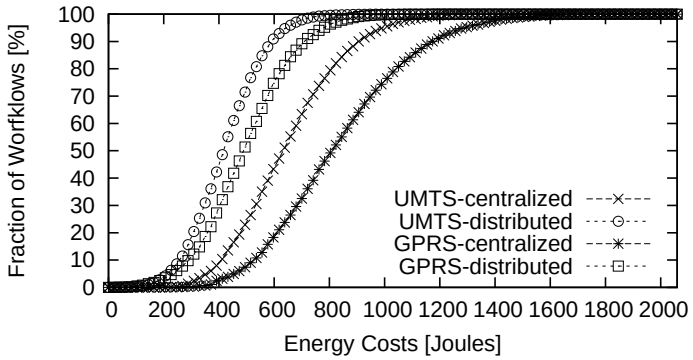


Figure 2.9: CDF for Absolute Energy Costs

2.4.5 Evaluation

In this section, we present our evaluation methodology and discuss the evaluation results. The goal of our evaluation is to give insight into the efficiency of our approach for a wide spectrum of different application scenarios. As there are no openly available data sets for real workflows, we rely on simulation and generate a large variety of different workflows for our evaluation.

The creation of the workflows is based on a grammar with rules to create sequential, conditional and parallel workflow structures. We apply the grammar to compose random workflow models from these partial structures, such that the number of activities per random workflow model ranges from 6 to 33. The varying size of workflows allows us to evaluate business processes of different complexity. The data flow defined by Θ_S is generated randomly according to a uniform distribution with a maximum of 5 MB per data link to allow for a variety of communication patterns. The values for Θ_A are implicitly defined by Θ_S to guarantee a consistent data flow in the workflow. That is, all data received by an activity is sent via its outgoing data flow links to other activities. Each activity is randomly assigned to either a human, a movable service or an infrastructure service. This assignment follows a uniform distribution. In order to assess the quality of our approach, we evaluate the energy savings per device and assume that a single human is interacting with the workflow using a single device.

We leverage on existing energy models from the area of pervasive computing to determine the costs related to data communication and service execution. We assume that the communication is done via a wide area wireless network since such networks are globally available and provide a maximum degree of mobility for users. Therefore, we studied the energy costs for wireless communication over GPRS and UMTS based on the energy models proposed by Balasubramanian et al. [10]. We derive the drained energy based on the size of the data to be transmitted. We also assume a maximum tail time in between successive wireless connections due to the execution of time-consuming human services.

Cuervo et al. determined the energy cost for executing a given piece of code based on profiling [24]. We employ their model to derive the service execution costs. This allows a workflow designer to predict the typical energy usage of services running on a mobile device. In our experiments, we assume that the execution of a service on the mobile device consumes a random amount of energy that is drawn from a uniform distribution with a minimum of 20J and maximum of 150J.

In our evaluation, we compare the energy consumption of two different deployment scenarios. The *distributed* placement is determined according to our distribution algorithm presented in Section 2.4.4. In contrast, the *centralized* placement refers to the classical deployment scenario where the entire workflow is executed in the infrastructure. We ran 10000 different simulation experiments and measured in

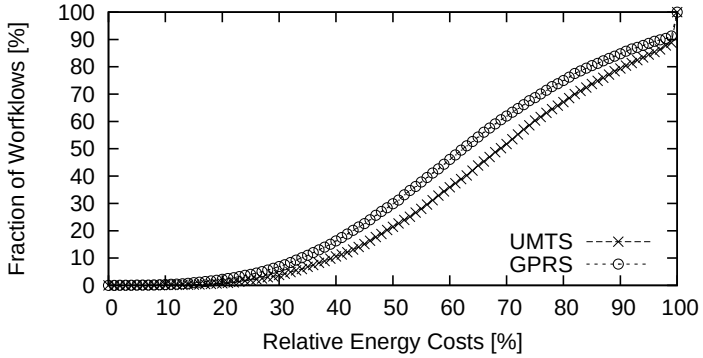


Figure 2.10: CDF for Relative Energy Costs

each experiment the required energy to execute the workflow with our distributed approach and with the centralized approach. Figure 2.9 shows the cumulative distribution function for the absolute energy costs, i.e., the fraction of all workflows which fall below a given energy budget. For communication over UMTS, 80% of all workflows consume less than 530J in case of our distributed approach. In contrast, only 29% of the workflows fall below this limit in case of a centralized deployment. The figures also demonstrates that 95% of all workflows do not exceed energy costs of 640J for our distributed approach, while the centralized approach requires energy up to 990J for the same fraction of workflows. We can also observe higher energy costs for both approaches when GPRS is used, since transmitting larger chunks of data consumes more energy due to the more limited bandwidth. For communication over GPRS, our approach guarantees that 80% of all workflows consume less or equal than 640J. However, only 23% of the workflows can meet the same energy constraints for the centralized execution.

Figure 2.10 shows the relative energy costs of our approach, measured as the fraction of energy consumed by the centralized approach. The figure depicts the cumulative distribution function, showing the fraction of workflows which remain below the given relative energy costs. While the performance for both GPRS and UMTS is similar, we can again observe slightly higher energy savings for GPRS-based scenarios due to the reasons explained above. In the best 10% of the cases, we have to spend less than 34% for GPRS and 40% for UMTS of the centralized energy consumption. Most of the workflows consume between 40% and 70% of the centralized energy consumption, depending on the degree of human interaction involved. On average, the workflow execution requires 63% of the centralized energy consumption for GPRS and 68% for UMTS. This represents a large improvement in energy usage, extending the lifetime of mobile devices significantly and reducing the costs of the underlying business processes.

2.5 Summary and Conclusions

In this chapter, we have discussed the problems of human interaction times and energy-efficiency in pervasive flow-based systems. For each of the two challenges, we have presented an algorithm that distributes flows such that the system maximizes its utility.

In Section 2.3, we introduced an algorithm that minimizes human interaction time in workflow systems based on a list-scheduling approach for mapping activities to network domains. We compared our algorithm *LWAA w/ HC* to non-partitioned and greedy approaches and showed that it improves interaction time by up to 80%. Hence, *LWAA w/ HC* reduces the interaction time of humans with workflows significantly and, thus, increases the processing throughput considerably. This can result in competitive advantages in a business environment. Additionally, it helps opening areas like pervasive computing for workflow technologies since it renders workflow technology less obtrusive.

In Section 2.4, we have presented an approach for distributing workflows among a set of mobile and infrastructure-based hosts in order to minimize the energy drained on mobile devices. We have developed a cost graph that represents the energy consumed by the execution of workflows. Based on this cost graph, we have presented an optimal minimum-cut-based algorithm for the energy-efficient placement of workflow activities. Our evaluation shows that our algorithm saves on average 37% of the energy for GPRS and 32% for UMTS over a purely infrastructure-based approach. In application domains that are heavily workflow-driven (like logistics and health care) this represents a significant energy saving that allows for longer operation between two recharge cycles and thus for less distractions in the daily routine of the involved personnel. In scenarios where the available battery capacities already ensure distraction-free processes, our approach allows for downgrading to cheaper technology with less capacity. Both routes lead to significant monetary savings in areas where mobile devices are indispensable tools.

Thus, our work represents an important step toward the cost-efficient and seamless integration of business processes and pervasive computing, enabling much more flexible workflow-driven applications that involve mobile users.

Chapter 3

Robust Flow Navigation

As explained in Section 1.1 of the introduction, our concepts for achieving robust flow navigation are twofold:

FlowCon and FlexCon We extract additional information about the likelihood for the occurrence of specific context events from the flow knowledge (the temporal structure and the state of a flow that is being executed). We employ Bayesian Network technology for building a model of this likelihood over time and extract the additional information from this model. We have built two systems for achieving that based on different flow models. FlowCon can handle *imperative flows*. These are flows that have a rather rigid structure where the sequence of activities is mostly well-defined through *transitions* (directed edges between the activities). This type of flows is easier to exploit for achieving robust navigation since the rigid structure encodes a lot of information. However, it also limits the freedom of the user executing the flow since it prescribes her actions. Therefore, we have investigated more flexible flow models in the next phase of our research. FlexCon can handle what we call *hybrid flow models*. These are models where activities may also be related through *constraints* that introduce a much loser coupling between activities. For example, a constraint may define that the associated activities must not be executed in the same flow, or it could specify that a certain activity must eventually be executed without specifying its exact position in the flow. These hybrid flows offer much more flexibility to the user. However, they also potentially offer less information for flow navigation. FlexCon deals with this problem by using Dynamic Bayesian Networks to account for the changing execution of the flow.

An important metric for the evaluation of FlowCon and FlexCon is the completion ratio of flows. This is the ratio of flows that is completed without getting stuck or ending in an error due to false navigation decisions. Without any assistance, a flow system that is faced with realistic uncertainty levels and noise only manages to complete up to 10% of all flows as our evaluations will show. With FlowCon, we observe a completion ratio or up to 90%. FlexCon performs worse due to the increased flexibility and dynamics in flow execution. But it still achieves between 20% and 80% flow completion.

FeVA Even if we manage to reduce the uncertainty of incoming events with FlowCon and FlexCon, there could still be errors in the sequence of events. Events could be missing, they could be arbitrarily added (both through recognition errors in the context management system), or they could simply arrive in the wrong order (due to timing issues). We have developed FeVA, as system that exploits the flow knowledge and uses fuzzy set theory to correct such errors as far as possible. On average FeVA assigns 91% of the context events correctly, and the number of successfully completed flows increases by 52% over a reference system without FEvA.

Overview of the Chapter The rest of this chapter is structured as follows: In Section 3.1, we will first present the real-world case study and experiments we conducted with our project partners in order to gather the necessary data for the evaluation of our systems. In Section 3.2, we take a look on the

related work in the area of flow navigation. Section 3.3 gives a number of basic definitions on which the following chapters are built. In Section 3.4, we take a closer look at the basic concepts employed in FlowCon and FlexCon before we present these systems in Sections 3.5 and 3.6. In Section 3.7, we present FeVA before we finish this chapter with a summary and conclusions in Section 3.8.

3.1 Real-World Scenario and Case Study

The application scenario we use to evaluate our approach to robust flow navigation is from the health care domain. We studied the processes conducted by nurses in a geriatric ward in Mainkofen, Germany over a period of 14 days. The ward is an intensive care station for elderly people suffering from dementia and similar old-age diseases. Each of the patients there needs help around-the-clock. There are well-defined medical guidelines for accomplishing the daily work within the ward. This scenario provides a relatively limited and basically fixed set of nurses and patients and the process structure is also quite stable. All activities performed (e.g. treatment, medication) stringently have to follow the guidelines and the results of some must be documented. But, there is rich human interaction between nurses and patients.

The traces we obtained from the nursing ward follow the daily morning routine of a single nurse. Each nurse was given a mobile phone which she wears in her coat pocket for data collection. The sensor readings available in the traces are (1) received WiFi signal strength, (2) measured magnetic field strength, (3) measured acceleration and (4) recorded sound The WiFi readings were used to estimate the indoor position of the nurse on a room-level granularity, the magnetic field sensor for facing direction. The necessary WiFi infrastructure was already present in the hospital, so there was no need to deploy further infrastructure. The acceleration data were used to do activity recognition like mode of locomotion. The recorded sound snippets were also used to classify activities according to typical background noises like the sound of a shower when a nurse is helping a patient taking a shower. For more complex context information, multiple of the mentioned modalities were used for recognition. The collected data has been manually labeled for a training set, but there is also an unlabeled test set.

Each trace covers about 2 and a half hours, where the nurse had to care for a total of three to five patients. The basic support for every patient is very similar and consists of four distinct steps. The (1) morning examination includes measuring the pulse and the blood pressure of the patient. Blood samples are taken regularly once or twice a week per patient. During the (2) morning hygiene, the nurse helps the patient with getting up, washing and dressing. Following that the nurses help the patients having their (3) breakfast. Finally she supervises and assists the patient taking his (4) daily morning medication according to the patients capabilities.

Through a mining process, we extracted workflows from the observations made at the ward. The respective processes have not been defined as workflows before. However, in this highly structured working environment, workflows are implicitly followed in order to fulfill a number of standards in terms of patient care. In total we collected 32 datasets from 15 different nurses, where each dataset covers the care of 3 to 5 patients, yielding a total of 130 observed workflow executions.

The purpose of applying a system of adaptable pervasive flows in this institution is twofold: First, the activities shall be automatically documented for the records for quality control, process improvement, and legal reasons. Second, the flow system shall give guidance in case the standard procedures are not followed in order to avoid mistakes and help inexperienced personnel in learning the procedures.

3.2 Related Work

In the following, we will investigate the state-of-the-art in the relevant areas of research. We will first discuss activity recognition systems and how they deal with context uncertainties. While our work is not directly associated with this area, it does provide a new approach for handling the uncertainties perceived in activity recognition systems on a higher layer by exploiting application knowledge. Subsequently, we

will take a closer look at the field of context-aware workflows. As our approach is based on flows, we investigate existing work on context-aware and mobile workflow management. Furthermore, we discuss the relation of FlowCon to the area of workflow mining and the handling of fuzzy or uncertain (context) information in workflow management systems.

There have been numerous studies on activity recognition in the health-care domain [12, 56, 16]. The major factors for decreasing the uncertainty in the recognition results are the selection of appropriate sensors and exploiting available application models. Biswas et al. [16] specifically remark that the recognition process can benefit from the knowledge of domain experts. A flow is a very detailed representation of expert application knowledge, that FlexCon uses to increase the accuracy of events.

Barger et al. [12] studied a health status monitoring application that learns behavioral patterns of a user from his daily activities using a number of motion sensors. But their system lacks an application model too, leading to missed events and false positives and a rather low recognition accuracy for uncommon situations.

Najafi et al. [56] have built a monitoring system for elderly people using one acceleration sensor, and detecting position transitions and mode of locomotion. While this approach performs very well for single transitions in a specific test scenario, the sensing quality decreases over extended periods of time due to the lack of an application model.

The presented approaches all use sophisticated activity recognition techniques, but do not consider the kind of application knowledge a flow provides, thus, neglecting the huge potential.

The integration of context information into classic workflows used in enterprises has first been suggested by Wieland et al. [76], who provide interfaces for accessing context information from within a workflow. This approach was later extended to deal with Quality of Context [75]. Here, a policy language is used to define the acceptable amount of uncertainty in context information and to filter out information that does not match the specified criteria. This approach is based on the idea to simply prevent the workflow from receiving uncertain information. However, if a workflow does not receive the information at all, this can be just as detrimental as receiving false information. We go one important step further by improving the information such that it becomes useful for the flow.

Mobile workflow execution is a key technology to enable the use of flows on mobile devices. A feasible approach to execute workflows on mobile devices has been presented by Hackmann et al. [34, 33]. But the mobile workflows considered do not take context information into account and thus they are suited for classical workflows only.

The PerFlows presented by Urbanski et al. [71] are context-aware, suitable for pervasive scenarios and provide flexible activity scheduling and processing. However, they require heavy user interaction to work properly. In our previous work [78], we presented an approach for dynamic context-awareness suited for pervasive flow-based applications. Both approaches neglect the handling of uncertain context information.

Workflow Mining comes in two different flavors. On the one hand, a new workflow is created from event logs of different applications in order to visualize the actual flow of work and possibly automate the created workflow using classical workflow management techniques. On the other hand existing workflows can be used to extract knowledge. While we also extract knowledge from flows, there are no approaches that improve context processing this way. Buffett and Geng[19] have proposed to label the activities of workflows by learning from event logs. This approach is somewhat similar to ours. It uses learning with Bayesian Networks, resolves the ordering of activities in the generated workflows and analyzes the paths taken in workflow execution. However the algorithm is applied to collected event logs and the workflows are mined in with an offline algorithm. Furthermore, it assumes that the log data contains no uncertain information, e.g., no real-world context is taken into account. In contrast, we know the flow structure and learn the event dependencies at runtime, taking uncertain context information into account.

Fuzzy Workflows, have been presented by Adam et al. [2, 3] A Fuzzy Workflow is able to make decisions based on fuzzy input information. The input from different sources is fed into a fuzzy logic operator

within the flow and the result leads to a clear decision which is used to continue the workflow execution accordingly. However, these approaches are extensions for classical workflows, which are usually not flexible enough for execution in a mobile context-aware environments. But even more importantly, they do not enable the workflow to contribute in decreasing uncertainty of context information.

There are plenty of workflow models based on petri-nets (e.g. [72]), and also fuzzy petri net variants have been proposed [60] and applied to workflows [64]. Basically all elements of a petri net – places, markers, and transitions – can be fuzzified and integrated into a fuzzy reasoning process. The fuzzified markers bear a similarity with our weighted event instances. However, we avoid having a fuzzy execution state, whose semantics are difficult to define, while we still allow soft decisions when navigating the flow as more context events become available.

In summary, classical context-aware workflows and workflows tailored for pervasive computing provide little mechanisms to deal with uncertain context information. Furthermore no system discussed here uses the knowledge encoded in the workflow to improve the accuracy of context recognition, thus reducing the amount of uncertainty the flow actually has to deal with.

3.3 Basic Definitions

In the following, we provide basic definitions of how context and flows are represented in our system. This is the foundation for the concepts introduced later in Sections 3.5, 3.6 and 3.7. We start by defining the context model.

3.3.1 Context Model

As the flows should not obstruct users in their daily routine, they are solely driven by *context events*. That is, a *Context Management System* (CMS) measures and detects events in the real world (e.g. the nurse entering a specific room) and provides these context events to the flow system. Therefore, adequate sensors and an activity recognition and context management system (CMS) must be available to gather context information and provide these context events. However, state of the art activity recognition systems have some drawbacks. Either, they require the precise deployment of (expensive) sensors, or the setup and training of the system is tedious. Cheaper activity recognition systems, e.g. based on standard smart phones, only provide moderate recognition rates, at best. While the former technology might be applicable in high-cost environments such as an operating room, we have to rely on the latter kind in most real-world situations.

In the scope of our scenarios, we assume that in practice the set of possible types of events is finite.

Definition 3.3.1 (Event Type) *A type of situation that can be recognized in the real world is referred to as an* event type $u \in U$*, where* U *denotes the universe of all event types that the CMS can measure.*

An event type describes the abstract semantics of an context event. For example, *nurse walking* could be an event type. Events of this type are created whenever a nurse changes her mode of locomotion to *walking*. Event types that represent semantically similar context can belong to a common *event type set*, and each event type belongs to at least one event type set.

Definition 3.3.2 (Event Type Set) *An event type set* $E \subset U$ *contains a number of event types* $E := \{u_1, \ldots, u_m\}, m > 0$*. A single event type can be a member of different event type sets.*

The event type set containing all event types for a nurse's locomotion modes could be, e.g. $E_\alpha = \{$*nurse walking*, *nurse sitting*, *nurse standing*$\}$. The purpose of an event type set is twofold: First, it allows the flow modeler to simply select the most appropriate context the activity should respond to. As seen below, a flow model defines a function that maps every activity to a number of distinct event type sets. Second, the related semantics of all event types in an event type set allows for a more accurate

recognition: Event types that are not contained in one of the expected event type sets of the current flow activity are likely to be out of scope. When executed the flow registers the event type sets of a running activity at the CMS and receives *event instances*.

Definition 3.3.3 (Event Instance) *An event instance $e \in U_e$ is an instance of a specific event type $u \in U$. U_e is the universe of all event instances occurring in the system. e belongs to a specific event type $u \in E$, and the uncertainty about which exact event type in E e belongs to is given by a probability distribution $I_E^e : E \mapsto [0, 1]$, where $\sum_{u \in E} I_E^e(u) = 1$.*

I_E^e is our basic model of uncertainty. Instead of saying that an event instance is of type u, the CMS provides the distribution I_E^e, and $I_E^e(u)$ is the probability that e is of type $u \in E$. For example if $u = nursewalking$ and $u \in E$ then $I_E^e(nursewalking) = 0.52$ indicates that the probability of e being of type *nurse walking* is 52%.

Definition 3.3.4 (Event Sequence) *Let $\mathcal{E} := \{E_1, \ldots, E_j\}$ be the set set of all event type sets used for a flow model. An Event Sequence $S = (e_1, \ldots, e_k)$ is an ordered list of k event instances, where each $e_i \in \mathcal{E}$.*

An event sequence S represents the context events and the temporal order in which they occur.

3.3.2 Flow Models

A *flow model* is a template for a specific type of flow. A runnable instance of such a model must be created whenever a flow is to be executed. We call this a *flow instance*. In the following, we also refer to such an instance simply as a *flow*. The flow instance is executed by a flow engine. In our work we consider two types of flow models:

- *Imperative flow models* are directed acyclic graphs with activities as vertexes and transitions as edges. The application programmer defines all the *activities* and their partial ordering using *transitions*. *Conditions*, that are annotated to the transitions, further influence the ordering.
- *Hybrid flow models* that contain *transitions* as well as *constraints* between activities and, thus, are a mixture of classical imperative production workflows [50] and declarative flexible [61] workflows. Transitions are annotated with boolean *conditions* over the possible set of context events while *constraints* consist of *linear temporal logic* expressions that describe the acceptable temporal relation of two or more tasks (e.g. *a* must be executed before *b*). If a flow modeler currently wants to use a mixture of both modeling paradigms he is required to add this flexibility in a hierarchical way [1]. He must decompose the application into a number of hierarchical layers, usually representing a different level of abstraction and choose the best modeling paradigm for each layer. Our hybrid flow model, allows the use of both paradigms directly on all abstraction levels and can also be applied to applications where the hierarchical decomposition is not possible or introduces further complexity.

We will define both types of models formally below.

Definition 3.3.5 (Imperative Flow Model) *An imperative flow model $\mathcal{F}$ is a 4-tuple $\mathcal{F} := (A, T, C, \mu)$, consisting of a set of activities A, a set of transitions T, a set of conditions C and a transition marker function μ.*

An imperative flow f is instantiated from a *flow model* $\mathcal{F}$ created by a programmer. $\mathcal{F}$ consists of a directed acyclic graph $G = (A, T)$ with activities $a \in A$ as nodes and directed transitions $t \in T \subset A \times A$ as edges. Each transition $t = (a_x, a_y)$ can be annotated with a logical condition c that depends on the context events received by the source activity a_x. If c evaluates to *true*, the flow makes the transition

from a_x to a_y. Some activities are mandatory, and must be completed for a successful flow execution. $\mathcal{F}$ acts as a template for an application (in our example documenting the work of a nurse). A *flow instance* is created at runtime (e.g. for a specific nurse) and executed on a *flow engine* that receives context events of a specific *context event type* from a CMS.

Definition 3.3.6 (Transition marker) *The transition marker function $\mu := T \rightarrow [true, false]$ assigns markers to all transitions in an imperative flow, where $\mu(t) = true$. If a transition has a marker, the execution of this transition is not required to be active in order to start the target activity of the transition.*

The transition markers allow joins of multiple flow branches where not all branches must or can be executed during a single flow execution. This way the execution of the activity is possible, when at least one of the previous activities has been completed. In flow diagrams, the markers are denoted as dots at the origin of transitions.

Definition 3.3.7 (Hybrid Flow Model) *A hybrid flow model $\mathcal{F}$ is a 4-tuple $\mathcal{F} := (A, T, C, L)$, consisting of a set of activities A, a set of transitions T, a set of conditions C, and a set of constraints L.*

Definition 3.3.8 (Activity) *An activity a represents an atomic piece of work within a flow. This includes invoking web services, internal computations, notifying a human about a task, or receiving context events indicating changes in the real world. The set $A := \{a_1, \ldots, a_n\}$ defines all activities of a flow. An arbitrary number of event types can be added to each activity. Let $\epsilon_a : \mathbb{N} \mapsto \mathcal{P}(U)$ be the event type assignment function for a, where $\mathcal{P}(U)$ denotes the powerset over the universe of events types. Further, let k be the number of event types associated with a, then $\epsilon_a(i)$ yields the i-th event type for $i \leq k$, and $\emptyset$ for $i > k$. We write ϵ_a for short when referring to the set of all event type sets assigned to a. Furthermore activities may be marked as mandatory.*

In a hybrid flow, activities may be executed arbitrary often and in any order. A flow can successfully complete its execution when all mandatory activities have been executed at least once. Transitions and constraints limit this flexibility and impose structural ordering on the flow activities.

Definition 3.3.9 (Transition) *Given a set of activities A, the set of all transitions within a flow is $T \subseteq A \times A$. A transition $t = (a_x, a_y)$ represents a directed control flow dependency from a_x to a_y with $a_x, a_y \in A$. A transition is annotated with exactly one transition condition, that is referred to as $c(t)$. Further, we define $d_{in}(a_i) := |\{(a_x, a_y) \in T | a_i = a_y\}|$ and $d_{out}(a_i) := |\{(a_x, a_y) \in T | a_i = a_x\}|$ as degree of incoming and outgoing transitions for an activity.*

The transitions allow certain control flow variants: linear sequences, parallel branching, joins and combinations of those. Conditional decisions can be made taking the context conditions into account.

Definition 3.3.10 (Context Condition) *A context condition c is inductively defined as $c \rightarrow u | (c_1 \vee c_2) | (c_1 \wedge c_2) | \neg(c_1)$ with $u \in U$ and c_1, c_2 are already valid conditions and the common semantics for the probabilistic logical operators.*

The condition $c(t)$ for $t = (a_x, a_y)$ is evaluated when a_x has received an event instance e for every ϵ_a. We insert the received event instances and check $c[u/I_E^e(u)] \geq t_n$ against the navigation threshold t_n. If the equation is fulfilled, the condition evaluates to true and the activity a_y is executed.

Definition 3.3.11 (Constraint) *A constraint l is an expression in linear temporal logic (LTL) that defines the temporal ordering of one or more activities in the flow. l is inductively defined as $l \rightarrow a | (l_1 \vee l_2)$(logical or)$| (l_1 \wedge l_2)$ (logical and)$| \neg(l_1)$ (logical negation)$| (l_1 \rightarrow l_2)$ (logical implication)$| \diamond (l_1)$ (eventually)$| \Box(l_1)$(globally)$| l_1 U l_2$(strong until), where $a \in A$ and l_1, l_2 are already valid constraints. The literals given in the expression l denote the completion of the respective activity a in the flow.*

Constraints are only used in hybrid flows and can be grouped in different classes such as existence, (negative) relation, (negative) order [61] and provided in a graphical representation (c.f. Figure 3.4). At runtime they are converted to final state machines (FSM) [30] and can be checked online for violations. If the FSM is in an accepting state the constraint is *valid*. When the FSM is not in an accepting state the constraint is *temporary violated*. The subsequent execution of further activities can eventually lead to a valid constraint. A constraint is permanently violated if the FSM reaches an error state and no sequence of activities can fulfill the constraint anymore. A flow can successfully complete its execution iff all constraints are valid.

The execution of the flow model yields a flow trace. When an activity is completed, this is recorded in the flow trace along with the event instances it received.

Definition 3.3.12 (Flow Trace) *A flow trace $\mathcal{T}$ is a sequence of completed activities $\mathcal{T} := (a_1, \ldots, a_j)$ in ascending order of completion times. The event instances each activity has received are also stored within the trace. Let $\theta(\mathcal{T}, a, u) \mapsto e$ be a function that yields the event instance $e \in u$ associated with activity a in trace $\mathcal{T}$.*

From a single trace, it is possible to reconstruct the actual execution of a flow instance and which context information, i.e. event instances, lead to this execution. All traces are stored in a flow history documenting the executions for later analysis. We use the flow history of a flow model as the data set for training probabilistic data structures in our algorithms.

3.4 The Basic Concept of FlowCon and FlexCon

Both, FlowCon and FlexCon, are based on the same principle. In the following, we will give an overview of this principle where we do not discriminate between the two. We will simply refer to them as ∗Con.

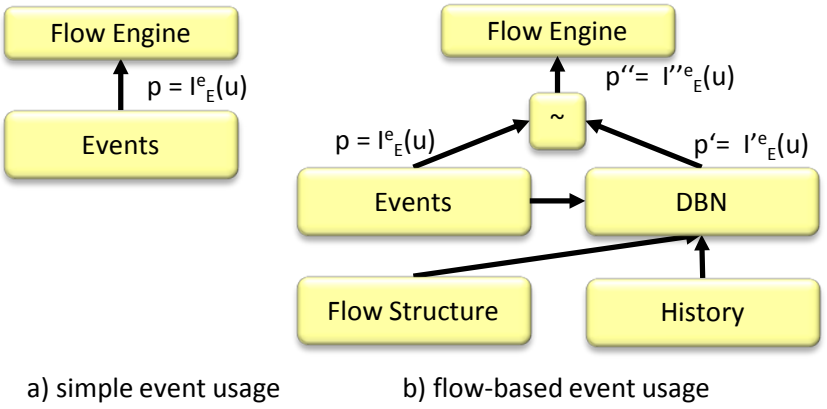


Figure 3.1: Architecture overview

The goal of ∗Con is to decrease the uncertainty of an event instance e. I.e. if e is of type u, then ∗Con shall collect additional evidence for this fact and increase the probability $p = I^e_E(u)$ for the event type u in the given distribution. To achieve this we use the flow as additional source of information. The flow model provides information concerning the structure (activities, transitions, constraints) of the flow and, thus, about the expected temporal relation of respective context events. The flow instance provides information given by its execution state, i.e. the current state of the activities and the already received context events.

Let us assume that the flow engine has started the execution of an activity a_1, and receives the event of the types associated with a_1, including E_α (c.f. Section 3.3.2). In a system without ∗Con, the flow engine would simply compare the probability $p = I^e_{E_\alpha}(u)$ with the engine's *navigation threshold* t_n and execute the respective transition if $p > t_n$. This simple approach is depicted in Figure 3.1 on the left.

∗Con, in the other hand, uses the information encoded in the flow model and the flow instance to infer additional evidence for the fact that e is actually of type u. Thus, it improves the probability distribution $I^e_{E_\alpha}$ that is the basis for the threshold comparison, leading to more reliable decisions and, thus, a more robust flow navigation.

∗Con uses *Bayesian Networks* (BN) to interpret context events depending on the current state of the flow. A BN is a probabilistic data structure that is flexible enough to represent the current flow state, the already received events, and the relation between the events according to the transitions and constraints of the flow model. ∗Con builds the structure of the BN from the flow model and trains the BN using traces of previously executed flows. This is shown in Figure 3.1 on the lower right. We explain the details of the construction algorithm in the following sections.

When a flow instance is executed, every incoming context event e is sent to the BN. Any such event is associated with a probability distribution I^e_E (cf. Definition 3.3.3). The BN infers an *additional* conditional probability distribution I'^e_E for e over E. The distribution I^e_E given by the CMS and I'^e_E given by the BN are combined, yielding an overall distribution $I''^e_{E_\alpha}$ which is then used by the flow engine to make its navigation decision. Our evaluations show that if $e \in u$ then, on average, $I''^e_E(u) > I^e_E(u)$. Hence, ∗Con reduces the uncertainty contained in the original distribution such that the flow engine can make more correct threshold decisions.

FlowCon was designed to operate on imperative flows where the ordering relations between activities are given by transitions, leaving little room for flexible execution. In this case, we use a *normal* Bayesian Network.

Flexcon, on the other hand, was built to handle hybrid flows that also contain constraints and, thus, allow for much more freedom and flexibility in the flow execution. To deal with this flexibility, FlexCon uses Dynamic Bayesian Networks (DBN) that can handle dynamically changing dependencies between context events. Using exact inference to get I'^e_E from a complex DBN, is computationally infeasible. Therefore, we use an approach based on *particle filters* [65] to increase the performance. We adapted the standard particle filter approach to reduce the computational effort, which allows us to use more particles on a more sparse DBN network and achieve more accurate inference results. We present a detailed description of the inference algorithm in Section 3.6.3.

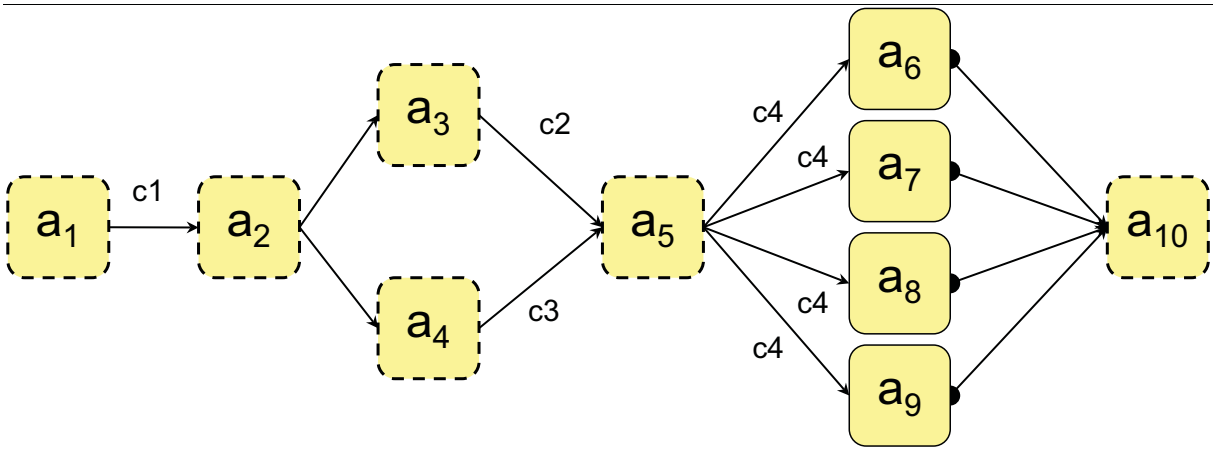
3.5 FlowCon - Robustness in Imperative Flows

3.5.1 Scenario Specificities

For the investigation of the FlowCon concepts, we use the case study described in Section 3.1 and regard a process within that study that consists of a number of concrete tasks. While most of these tasks are accomplished every day, each nurse flexibly alters the execution order. In order to limit these flexible changes, we focus the process execution on the blood sample examination process. In the following, we describe the process guideline in detail and point at the possible execution variations that may happen. Those variations are important for our algorithm design because the knowledge we extract is influenced by the habits of the nurse.

When a blood sample examination is scheduled for a patient—this is documented in the patient record—the nurse takes it after the daily measurement of pulse and blood pressure. A reusable butterfly needle is used, because there are taken up to four blood samples in a row. A formal representation of the flow is depicted in Figure 3.2. To obtain a blood sample the nurse has to perform the following activities. First, she (a_1) fastens a cuff to the upper arm of the patient. She then starts (a_2) searching a vein for setting the butterfly. After that, she (a_3) unpacks the butterfly and (a_4) disinfects the elbow pit. She punctures the patient (a_5) setting the butterfly and (a_6-a_9) takes the samples. Finally, she (a_{10}) labels each sample with the patients credentials.

While getting the blood sample from the patient, the nurse basically has two variation options. She can either disinfect the elbow pit first and then unpack the butterfly, or the other way around. However



a_1 apply Cuff	a_3 unpack butterfly	a_5 set butterfly	a_7 sample 2	a_9 sample 4
a_2 search vein	a_4 disinfect elbow pit	a_6 sample 1	a_8 sample 3	a_{10} labeling

Figure 3.2: Blood Sample Flow

she must complete both activities before she can set the butterfly. Moreover, she is free to chose the order in which the individual samples are taken when she has set the butterfly.

The mentioned variations lead to interesting questions. When the activity a_2—search vein—has been recognized, the context system cannot know which will be the next activity that the nurse executes. Because of this, it is much harder to recognize the following activity compared to a scenario with a predefined fixed sequence of activities. In this case FlowCon is able to increase context recognition performance. Furthermore, when the flow execution waits for an activity a_5—set butterfly—which depends on more than one previous activity, the recognition of the preceding activities (a_3, a_4) increases the probability that the next recognized activity will likely be a_5.

The correct flow execution leverages automatic documentation of the blood sample taking and relives the nurse of some of the paperwork. But this is a tough task, because it requires to recognize the activities for every single step just using the uncertain activity recognition results to drive the workflow execution.

3.5.2 Algorithm

The general goal of FlowCon is to increase the accuracy of the events the flow execution relies on. More specifically, we adjust the probability distribution of an event instance, so that the statistically most probable event type will be favored. The probability of this event type is increased, while simultaneously the probabilities of the possible other events types are decreased. This way the accuracy of the event types expected by the application will be increased and their uncertainty is decreased. We train a Bayesian Network (BN)[65] to extract the knowledge from the flow, which event type currently is the most probable. BN training consists of two phases: structure learning and parameter learning. While parameter learning can be achieved efficiently from a set of given observations, learning the optimal structure of a BN from such data is NP-hard. The FlowCon algorithm avoids the structure learning problem completely. It basically works in three steps. First, it analyzes the flow structure and generates a BN structure from that analysis. In the second step, it then uses the observed data from the flow history to do the parameter learning. While both of these steps can happen offline before the actual instantiation of a certain flow, the third step—the information combination—is executed at runtime. A comparison to the naive approach is depicted in Figure 3.1. Usually the application has to deal with the provided probability of an event type (e_1, p), but FlowCon queries the BN for the current event type, using the actual execution state of the flow and the already received event instances. It then combines the event type with its derived sta-

tistical probability from the BN (e_1, p') and generates a new probability for the event type (e_1, p''). The probability p'' will be higher than p if e_1 is statistically more probable for the application in the given context.

3.5.2.1 Structure Learning

To build the structure of the BN from the flow structure we assume that there exists a dependency between the events associated to two activities a_x and a_y if there exists a transition $t = (a_x, a_y)$. For example, in terms of our blood sample flow, the occurrence of the event type e_1—apply cuff—necessitates the occurrence of the event type e_2—search vein—afterwards. While this dependency is simple, there are more difficult cases. When we consider forks $d_{out}(a_2) = 2$, it is not clear if there is a dependency between the event types of a_2 and the event types of a_3, a_4. A single nurse could disinfect the arm first every time and then unpack the butterfly. This does of course change the statistical dependency between the respective event types. The one between e_2—search vein—and e_3—unpack butterfly— will be low, while the other between e_2 and e_4—disinfect elbow pit— will be high. When we further consider the activities with $d_{in}(a) > 1$ we see that the occurrence of an event type may depended on more than one previous event. Some of the preceding events may have a strong statistical dependency, while others have no effect at all. However, we create each of the possible dependencies in the beginning and adapt their strength later in the parameter learning phase, e.g. to deal with changing behavior of nurses.

We use a Bayesian Network $BN := (N, D)$ to represent the statistical dependencies from the flows, where $n \in N$ is a node and $d \in D$ is a conditional dependency. The nodes of the BN represent discrete random variables. The state space of a node n is equal to the event type set E adding a `null` class. For every event type set of an activity we create a node $n = a.E$ identified by the name of the activity and the event type set. After that, we add the dependencies between those nodes where the activities also have a directed dependency, or more formally:

$$((a_x, a_y) \in T) \wedge (\exists i : \epsilon_{a_x}(i) = E) \wedge (\exists j : \epsilon_{a_y}(j)) = E \Rightarrow$$
$$((a_x.E), (a_y.E) \in D)$$

The result is a structured BN. Please note that the learning has been very simple, because we have the flow structure which provides us with a realistic assumption which event type sets are related to each other. Furthermore, the structure is fixed for a specific flow model. Therefore, this step can be performed offline right after the modeling phase. However, there may be multiple instances of the BN in use for different locations where the flow is actually deployed and executed, or for different actors that are associated with the flow. For example, there could be a single network trained for every nurse to take personal habits into account when executing the flow and processing the context information.

3.5.2.2 Parameter Learning

The next step is to train the BN with the actual statistical dependencies that have occurred. At first the conditional probability tables (CPT)s of each node are initialized with a uniform distribution. In order to train a BN we need a training data set with observations for the value of each node. We use the flow traces that are collected for each execution of a flow as data set for training. As the event instances are stored together with the flow trace, it can be converted to an observation of the values.

As an example, we look at a trace $\mathcal{T}$ from our blood sample flow. For activity a_1 there is an event instance $I_{E_b} = \theta_{a_1}(\epsilon_a(1))$ stored in the trace. The probability distribution of this event instance I_{E_b} indicates that e_1 is the most significant event, i.e. the one with the highest probability. So for training from this trace we would set the observed value for the corresponding node $a_1.E_b$ to e_1.

The event types associated to activities that have not been executed in a trace cannot provide event instances, thus the corresponding nodes are set to the to the `null` state for this trace. In the mentioned trace the activity a_7 may not have been executed because the corresponding blood test was not scheduled.

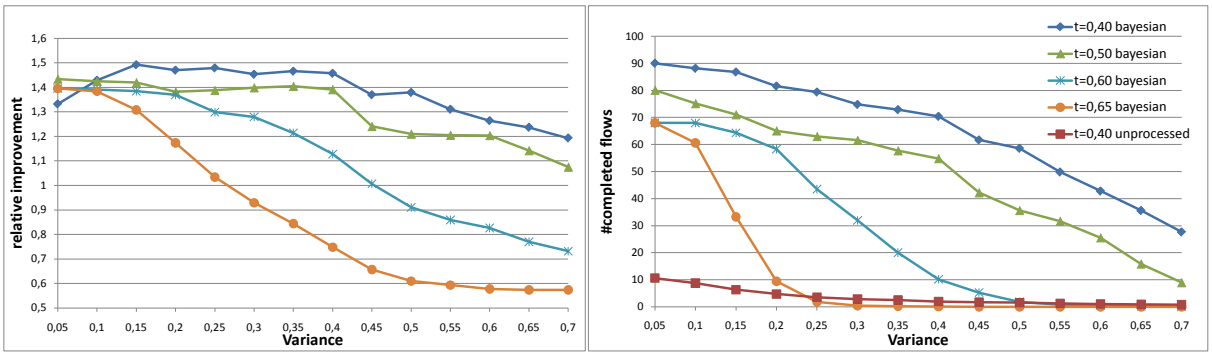


Figure 3.3: Simulation Results

So the value for the corresponding node $a_7.E_b$ would be `null`. The BN can be trained incrementally with the traces, as they become available. This way the CPTs are adjusted until the dependencies are appropriately represented. As we demonstrate in the evaluations a rather small training set of 25 to 50 traces is sufficient for training.

3.5.2.3 Information Combination

The third and final step is to retrieve the information from the BN and combine it with current context information to increase accuracy. Querying a BN usually means to initialize some of the variables with observed values and compute the conditional probabilities of the unknown variables. The observed values for the instance of the flow that is currently executed, are the past events that have already been recognized. We take all available events as evidence into account.

When the next event e arrives that only marginally denotes that the expected event type u has happened, we compute the conditional probability for this event type using the previous event instances as current observations. The result from the BN is also a probability distribution function for the event type set E. We combine this result with the probability distribution function I_E of the actual event instance. We compute the average probability for each event type in the distribution and normalize the results afterwards to make the probability distribution valid again. This averaging adjusts the probability of each event type by the amount of its statistical probability under the given context of the previously occurred events. Here we actually adjust the context measurement. The probability of an event type that statistically occurs more often is increased, while the other way around, the probability gets lowered. Note that higher probability also means a higher accuracy for the measured event type. The resulting probability distribution is stored in the event instance. The event instance is then sent to the flow instead of the original one, and processed according to the flow execution semantics.

3.5.3 Evaluation

To evaluate our system, we used the blood sample examination flow we presented in Figure 3.2. The scenario data and the flow were directly extracted from the collected real-world traces. Unfortunately, some of the event data had to be simulated and we discuss this in the setup section, along with some technical details. Following that, we present our evaluation results and analyze them.

3.5.3.1 Setup

Our simulation setup consists of three main components. First a basic flow engine that is able to execute our flow models using the JUNG Framework [41]. The flow engine defines the navigation threshold t_n as described in Section 3.3.2. The navigation threshold is our first simulation parameter. The flow engine also implements the evaluation semantics defined for the conditions based on Probability Theory.

The second component is the Bayesian Network Event Processor that implements our algorithm and processes the event instances. To represent the Bayesian Networks, we used the well-known Weka framework [35].

The third component is the context system which is responsible for feeding the context events into the flow engine. Based on the recognition results from the real traces, we generated artificial traces with the same structure and probability distributions in order to produce statistically relevant results. We first created lists of events that would allow the engine to complete the flow successfully. We did not consider out of order events or completely missing ones, but just the uncertainty of false recognitions. The event lists were then assigned with the results of the real recognition to keep the simulation as realistic as possible. The average recognition probabilities were between 40% and 60% for the correct events. But there have also been false recognitions with probabilities also up to 40%. We further added noise to those single probabilities, which effectively introduces a variance v on the absolute recognition probabilities. The resulting values were normalized to get a sound probability distribution. The variance value v is our second simulation parameter.

An experiment we simulated, consisted of the subsequent execution of 100 blood sample flows. Each experiment was repeated 25 times with the same parameter settings to achieve statistical relevance. We started with a freshly initialized BN every time and had no training data available from flow histories i.e. the parameter learning phase of our algorithm is performed online. The number of available traces for training then increases with every completed flow instance. We chose navigation thresholds between $t_n = 0.4$ and $t_n = 0.65$ with steps of 0.05 and accuracy variance values between $v = 0.05$ and $v = 0.7$ also in steps of 0.05. Please note that a variance value of $v = 0.4$ basically introduces the same amount of noise into the simulation as we have recognition probabilities from the traces. When we further increase the variance up to $v = 0.7$ this can be interpreted as feeding significantly more noise into the flow compared to the given recognition probabilities.

3.5.3.2 Results

We measured two properties of our system, the accuracy improvement of the events that should be delivered to the flow engine in order to allow a correct execution of the flow and the overall number of completed flows which can be interpreted as the robustness of the flow execution.

Event Improvement For the event improvement we measured the relative event improvement, which is depicted in Figure 3.3 on the left. By relative event improvement we mean the probability of the significant event of the event instance divided by its original probability before the processing with our algorithm (p''/p c.f. Figure 3.1). We left out some curves for better visibility. For the thresholds $t_n = 0.4$ and $t_n = 0.5$ we observe a very good accuracy improvement performance between 49% and 39%, for variance values up to $v = 0.4$. This conditions indicate a system that has equal to higher requirements for the recognition accuracies that could actually be provided. Furthermore we can deal with a significant amount of noise quite well. However when we further increase the variance up to $v = 0.7$ the average event improvement slowly degrades to only 7%. But we still manage to improve the recognition probabilities a little. When we further increase the navigation threshold $t_n = 0.6$ and $t_n = 0.65$ the performance degrades much faster. While FlowCon is still able to achieve a good improvement for small variance values up to $v = 0.15$, we quickly get counter productive results when we further increase the variance. The break even point, where we actually make things worse using FlowCon, is $v = 0.45$ for a threshold $t_n = 0.6$ and $v = 0.25$ for a threshold $t_n = 0.65$. This strong degradation can be explained as we train the BN online during the experiments. The correct training gets more difficult and finally impossible with higher variance, values because we have fewer correct traces and the navigation threshold to achieve a correct trace is very high.

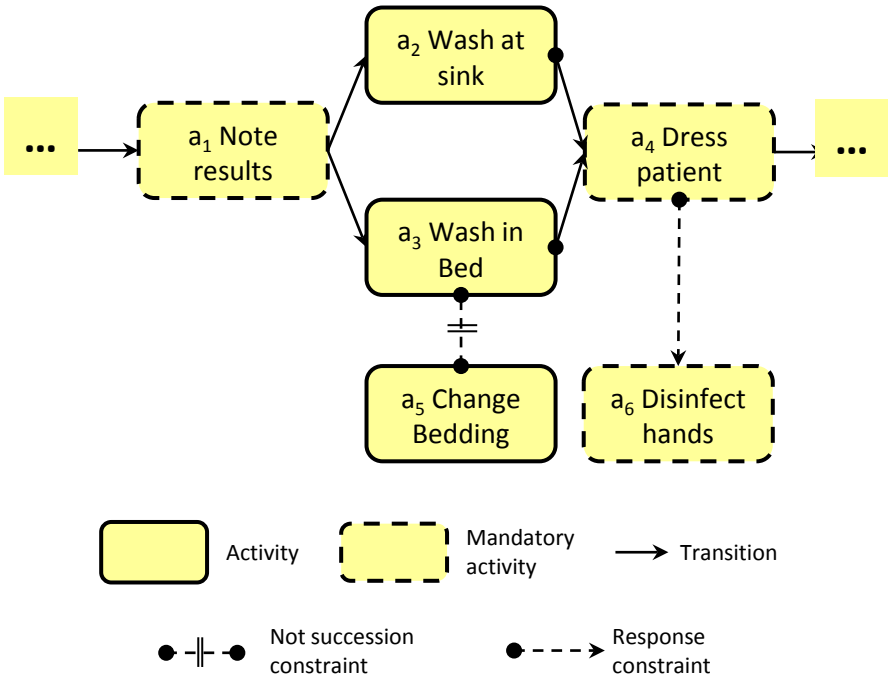


Figure 3.4: Morning hygiene flow

Flow Execution Robustness The second observed property is ratio of flows that were executed successfully during an experiment. Those results are depicted in Figure 3.3 on the right. As a reference, we have also shown here the performance of our flow engine under the same conditions, but without the processing accomplished by FlowCon. Given this setting, the engine is only able to complete an average between 10.6% and 0.8% of all flows, while the combination with the FlowCon algorithm yields an average number of completed flows between 90.0% and 27.6% which is a significant improvement. The same degradation behavior as in the event quality improvement can also be observed for the overall system robustness. We perform quite well with slow degradation and a measurable drop at a variance $v = 0.45$ for the lower thresholds ($t_n = 0.4$ and $t_n = 0.5$). For the upper thresholds, the performance degrades much faster for the reasons we explained before.

3.6 FlexCon - Robustness in Hybrid Flows

3.6.1 Scenario Specificities

In order to evaluate FlexCon, we also used the data from our hospital case study. However, this time we used hybrid flow models from the study. A typical flow, e.g. from the morning routine, consists of 30 to 50 activities of which about 20% have no strict order. The entire navigation in such a flow depends on context events (i.e. the correct next activity is chosen based on the context events received). An example fragment is depicted in Figure 3.4. Dashed boxes depict mandatory *activities* that need to be executed unconditionally while solid boxes are optional. For example, a_2 and a_3 are optional. The execution of a flow instance is valid if, one of them, both or non of them have been executed, while a_1, a_4, and a_6 need to be executed for the flow to be successful. Solid arrows are *transitions* that imply a strict ordering between the activities: a_1 must be followed by either a_2 or a_3 and a_4 must follow both a_1 and a_2. The dashed lines are *constraints* that define certain restrictions on the execution order of the related activities. The figure depicts two examples: the semantics of the *not succession* constraint between a_3 and a_5 is that a valid flow execution must not contain both activities. It may contain either one or none of them, and if one is executed, it can be executed arbitrarily often. As we explained in Section 3.3.2, constraints

can be arbitrary linear temporal logic expressions. Some of them have been translated into a graphical representation.

The flow shown in Figure 3.4 is a fragment of a larger flow that models the actual processes found in the Mainkofen nursing ward. Its overall semantics is the following: When a nurse arrives at this fragment, she must document the results (a_1) of the preceding steps, which include some regular morning examinations, such as measuring blood pressure. As these examinations are carried out without assistance of an electronic device, the flow ensures that the nurse will not forget the results during the following steps. Then she has to take a decision: she may wash the patient at the sink (a_2) or in his bed (a_3), depending on the patients condition and mood. In FlexCon actually both activities are entered as soon as a_1 has completed. Depending on the incoming context events, either one or both are executed. If the nurse decides to wash the patient in his bed (a_3), she cannot change the bedding (a_5) since the patient still never leaves his bed during the whole procedure (this is done in a different flow). After the nurse has completed the washing activity, she needs to dress the patient (a_4). When she dressed him, she must disinfect her hands (a_6) at some later point in time, possibly after a number of other intermediate activities. But, she may disinfect her hands at any point in time, while the flow is being executed. This is beneficial in two ways. First, the nurse can flexibly decide to disinfect her hands multiple times, e.g. during washing the patient, also allowing the system to keep track of her personal hygiene as well as the patients. Second, the flow can guide the nurse to disinfect her hands before she continues to care for another patient, this way enforcing the hospitals hygiene rules.

3.6.2 Dynamic Bayesian Network - Structure and Learning

A Bayesian Network $\mathcal{BN} = (\bar{X}, D)$ is a directed acyclic graph representing a joint probability distribution over a number of random variables (RVs) $\{X_1, ...X_n\} = \bar{X}$. $\bar{X}$ represents the nodes and the edges $D \subseteq \bar{X} \times \bar{X}$ define a conditional dependency from the source RV to the target RV. In FlowCon, we used BNs as the flows where based on imperative models that specify the complete execution order. Therefore, the simple static BNs were sufficient. The hybrid model in FlexCon, however, introduces much more freedom for the users to drive the flow forward in different ways and, thus, more dynamics. The static BN model does not support such a dynamically changing probabilistic process. Therefore, FlexCon employs Dynamics Bayesian Networks which are tailored for dynamically changing systems.

In a DBN [65, 55], the state of the RV changes over time and the observed values for the RV in the current *time slice* $\bar{X}_t$ depend on the observations of one or more previous time slices. This dependency is expressed by the *transition model* $\mathcal{TM} = P(\bar{X}_t|\bar{X}_{t-1})$. When we write $X_{1,0}$, we refer to the RV X_1 in the time slice $t = 0$. Additionally, a DBN has a prior distribution $\mathcal{PD} = P(\bar{X}_0)$ for time $t = 0$, such that the definition of a DBN is given as follows: $\mathcal{DBN} = (\bar{X}, \mathcal{TM}, \mathcal{PD})$[1].

3.6.2.1 DBN Construction

Let $\mathcal{F}_1 = (A, T, C, L)$ be the flow model from our example in Section 3.6.1. For each $a \in A$ and each $E \in \epsilon_a$, FlexCon creates a node in the DBN. More formally, the function $\chi : A \times \mathcal{P}(U) \rightarrow \bar{X}$ maps an activity a and an event type set E to a unique RV X of the DBN. Let further $\bar{\chi}(a, \epsilon_a)$ be the set of all RVs associated with activity a. $\chi(a, E) = X$ with $E \in \epsilon_a$ is discrete and can assume the same values present in the event type set E plus a *null* class, represented by $\perp$. For example, let us consider a_1 and $E_\alpha \in \epsilon_{a_1}$. The respective random variable $\chi(a_1, E_\alpha) = X_\alpha$ can assume any value from {*wash, dry, write, fetch, disinfect*, $\perp$}. $\chi(a, E)_t$ and $\bar{\chi}(a, \epsilon_a)_t$ refer to the respective RVs in time slice t.

The *time slices* in our DBN are defined with respect to the execution state of the flow: Every time an activity completes its execution and the flow state is changed accordingly, we enter the next time slice in the DBN. FlexCon creates the transition model (the time dependencies) from the transitions and constraints in the flow model. Both of them enforce an execution order on the set of activities. We map

[1] Since FlexCon has no hidden variables, there is no need for a sensor model as it is usually found in the DBN definition

these order relations to the transition model, introducing directed edges (dependencies) from one time slice to the next. The strength of these dependencies in learned from flow traces (past flow executions) in a subsequent step. In the following, we describe the construction and learning phases first for transitions and then for constraints.

A transition $t = (a_x, a_y) \in T$ between two activities represents a very strong dependency as a_y can only be executed when a_x has been completed. Therefore, we create a dependency in the network for a pair of RVs if a transition exists between the respective activities as follows.

$$(\chi(a_x, E_x)_t, \chi(a_y, E_y)_{t+1}) \in P(\bar{X}_{t+1}|\bar{X}_t) \iff ((a_x, a_y) \in T) \wedge (E_x \in \epsilon_{a_x}) \wedge (E_y \in \epsilon_{a_y}).$$

For example, consider the activities a_1 and a_2 in Figure 3.4: They have a transition and, therefore, each $X \in \bar{\chi}(a_2, \epsilon_{a_2})_{t+1}$ would have $\chi(a_1, E_\alpha)_t$ as parent node, because $E_\alpha \in \epsilon_{a_1}$.

As constraints usually provide a less strict ordering of activities it is more difficult to derive the correct dependencies for the transition model. These dependencies can be different for each execution trace of the same flow. Let $l_1 = \Box(a_3 \rightarrow \neg(\diamond(a_5)))$ represent the *not-succession* constraint in the example in Figure 3.4. First, FlexCon assumes that there is a bidirectional dependency between all the activities that are contained as literals in the expression (a_3 and a_5 in the example). Hence, FlexCon adds $(X_{3,t}, X_{5,t+1})$ and $(X_{5,t}, X_{3,t+1})$, with $X_3 \in \bar{\chi}(a_3, \epsilon_{a_3})$ and $X_5 \in \bar{\chi}(a_5, \epsilon_{a_5})$ as dependencies in the DBN. In a second step, FlexCon determines the type of dependency that has to be included in the transition model $\mathcal{TM}$. If the sequential execution of the originating activity a_3 and the the target activity a_5 of the dependency *permanently violates* the constraint (as is the case in the example), FlexCon marks this dependency as *negative*. Negative dependencies are handled differently in the learning process as described below. If the sequential execution leads to a *valid* or *temporarily violated* constraint (c.f. Section 3.3.2), the dependency is handled like a transition. If the subsequent execution of the two activities has no influence on the constraint, we do not add a dependency at all. The latter is the case for the *response* constraint between a_4 and a_6 in Figure 3.4, where the execution of a_6 has absolutely no dependency on the execution of a_4.

3.6.2.2 DBN Learning

In order to learn the strength of dependencies in the DBN, we use the flow history as training data, counting the occurrences of all event pairs and learning their joint probability distribution. The portion of the flow history that is relevant for the learning is controlled by a sliding window algorithm taking only a number of recent traces into account. This helps in controlling the effectiveness of the learning procedure in the face of a changing behavior of the flow system.

For dependencies originating from flow transitions, the simple counting algorithm as explained above is sufficient. For constraints, we have to apply a different mechanism: In order to learn the strength of negative relations, we increase the count of the *null*-class for every trace where no such event sequence could be observed. This leads to a reduced probability of any other event type of the respective event type set. As an example, consider the *not succession* constraint of a_3 and a_5 again. The execution of a_3 will indicate that a_5 is never going to happen in any valid execution of this flow instance. Therefore, we reduce the belief of the DBN that any of the events associated with a_5 is likely to be recognized. An inexperienced nurse may execute the activity sequence a_3, a_5 nonetheless, but the flow can provide guidance for this case, preventing the nurse from violating the constraint l_1.

3.6.2.3 DBN Initialization

Finally, we need to initialize the DBN for $t = 0$, and provide the prior distribution $\mathcal{PD} = P(\bar{X}_0)$. This distribution is also extracted from the flow history: We search for traces of the respective flow model and create individual distributions for all the activities the flow has been started with at least once. For $\mathcal{F}_1$, this includes a_1, a_5 and a_6, and the distribution for $E_\alpha \in \epsilon_{a_1}$ could have the following values: $P(wash) = 0.01$,

$P(dry) = 0.01$, $P(write) = 0.85$, $P(fetch) = 0.05$, $P(disinfect) = 0.01$ and $P(\perp) = 0.07$. In most of the cases the correct *writing* activity has been recorded. In some cases, *fetch* has been misinterpreted, while sometimes there was no meaningful evidence at all ($\perp$). The rates for the uncommon activities (*wash, dry, disinfect*) are even lower.

3.6.3 Clustered Particle Filtering

In order to exploit the knowledge encoded in the DBN for a specific flow model, a process called *inference* has to be executed. That is, the posteriori distribution of the variables (nodes) has to be calculated given real *evidence*. In our case, the evidence are the real context events received from the CMS in time slice t, and the inference is done by computing all the conditional probabilities for the variables in time slice $t+1$. Exact inference is infeasible for complex DBNs like the ones generated from flows. Even more so, as this process is running in parallel to the flow execution: Whenever new evidence is available, the inference has to be done to get the probability distributions for the upcoming context events. Therefore, FlexCon uses a heuristic approach that is based on particle filters [65]. That is, we use a large number of random samples (the particles) from the distribution of the DBN at a certain time slice t and propagate them through the DBN to approximate the individual distributions associated with each node in the following time slice of the DBN. A particle filter approximates the exact distribution by generating a set of particles $N(\bar{X})$ for all random variables. The higher the number of particles the better the approximation of the real distribution. But the computation time grows linearly with the number of particles.

To propagate and calculate probabilities in the DBN the filter executes the following four steps. To initialize the filter, it first generates an initial particle set $N(\bar{X}_0)$ sampled from the prior distribution $\mathcal{PD} = P(\bar{X}_0)$ given by the DBN. In a second step each particle is propagated to the next time slice ($t = 1$ in this case) according to the distribution given by the conditional probability table. In the third step, the particles are weighted with the evidence available at the current time slice. Each particle is multiplied with the probability of the current observation. In the final step, the set of particles is resampled according to the weight of the individual particles. A detailed description of the basic principles has been published by Russel and Norvig [65].

We modified this standard algorithm as explained in the following, to accommodate it to the needs of FlexCon. The result is a *clustered particle filter* that is similar to the F3 filter presented by Ng et al. [57]. First of all, a single particle in FlexCon does not represent a full sample of $\bar{X}$ but only a sample of a subset of the variables $\bigcup_{E \in \epsilon_a} \chi(a, E)$, i.e. all variables of a single activity. Therefore, we call it clustered particle filtering, where each cluster can also be identified by $N(\bar{\chi}(a, \epsilon_a))$. This is an useful abstraction for a number of reasons. Each time slice in the DBN covers the completion of a single activity in the flow. Therefore, it is enough to process particles of that activity. All other particles are only propagated as they may be needed later on. This allows us to increase the total number of particles as the average processing load per particle is decreased. The unprocessed particles can be directly transferred to the same node in the next time slice, without the need for a dependency between these nodes.

For example, consider the trace $\mathcal{T}_1 = (a_1, a_6, a_3, a_4, a_6)$. After executing a_1, the particles from $\bar{\chi}(a_1, \epsilon_{a_1})_0$ are propagated to $\bar{\chi}(a_3, \epsilon_{a_3})_1$ since there is a transition (a_1, a_3), while $\bar{\chi}(a_6, \epsilon_{a_6})_0$ are just passed to $\bar{\chi}(a_6, \epsilon_{a_6})_1$, without further processing.

The second modification changes the propagation and weighting steps. Usually the full set of evidence, i.e. $P(\bar{\chi}(a', \epsilon_{a'})_{t+1} | \bar{X}_t)$, is available for propagating the particles in time slice t. As we only process the particles for a single activity a and only observe the received events for this activity as evidence, we can only rely on the conditional probability $P(\bar{\chi}(a', \epsilon_{a'})_{t+1} | \bar{\chi}(a, \epsilon_a)_t)$, instead. This means that we cannot use the evidence of events that have been observed "outside" of the current cluster $N(\bar{\chi}(a, \epsilon_a)_t)$. As a consequence we introduce an small error in the inference. However, the majority of $X \in \bar{X}$ will be independent from the variables in $\bar{\chi}(a', \epsilon_{a'})$, because there is no dependency defined by the flow. Therefore the introduced error is rather low and we actually discuss in Section 3.7.3 that not using this evidence makes FlexCon a bit more robust. Alternatively, it would also be possible to *sample* the evidence from

the current distribution $N(\bar{X} \setminus \bar{\chi}(a, \epsilon_a))$ of the other activities, but this also introduces inference errors.

After the propagation phase, the actual observations (i.e. the received event instances) become available to the DBN. We can then weight the particles multiplying the number of particles $|N(\chi(a, E) = u)|$ for a specific event type u with the actual probability of the event type given by $I^e_E(u)$. Based on the computed weights all the particles for $\bar{\chi}(a, \epsilon_a)$ are resampled according to the distribution of the weighted particles.

The third modification is the actual processing of the received event instance e in order to decrease its uncertainty. This step is accomplished after the propagation of the particles and before the weighting. We compute the conditional probability weights for I'^e_E from the particles in $\chi(a, E)$, where the weight

$$p' = \frac{|N(\chi(a, E) = u)|}{|N(\chi(a, E))|}$$

for $I'^e_E(u)$ is just the relative particle frequency, as the distribution in the sample $N(\bar{\chi}(a, \epsilon_a))$ represents a sufficient approximation of the correct conditional probability distribution. All probabilities $p = I^e_E(u)$ are added to the respective p' and the resulting distribution is normalized again, yielding $I''^e_E(u)$

Algorithm 6 Clustered Particle Filter Algorithm

```
Input: DBN = (X̄, TM, PD), a, e[]
if N(X̄) = ∅ then
  N(χ̄(a, ε_a)) ← createInitialParticleSet(PD)
end if
for all e ∈ e[] do
  weightEvent(e, I^e_E, χ(a, E))
  weightParticles(N(χ(a, E)), I^e_E)
  N(χ(a, E)) ← resampleParticles(N(χ(a, E)))
end for
propagateParticles(N(χ̄(a, ε_a)), TM)
```

Algorithm 6 depicts the standard particle filter algorithm including the changes introduced by FlexCon. The input to the algorithm includes the $\mathcal{DBN}$, the currently completed activity a and the set of event instances $e[]$, a has received.

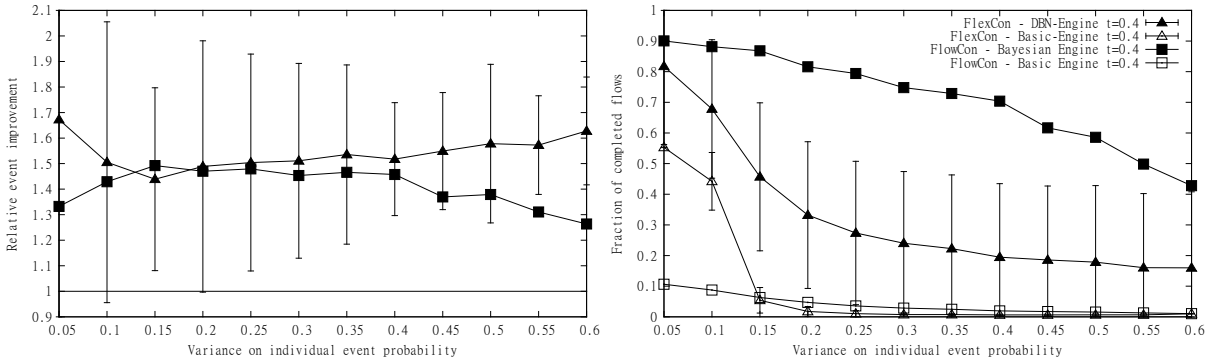
3.6.4 Evaluation

For our evaluation, we have generated flows according to a probabilistic pattern-based model [22] that has the same properties as the flows observed in the real-world hospital scenario. We do this to get a number of flows that is large enough to achieve statistical relevance. The flows we generate have the same average number of activities and the same structural properties. I.e. they have the same ratio between activities that have normal transitions and activities that are connected to other activities by constraints.

Use of *flow patterns* [48] allows us to generate imperative flows based on structures commonly found in human-centric flows. We generate these flows and randomly add a respective portion of unconnected *constraint-based activities* (CBAs) to the flow. Next, we randomly generate constraints and use these to connect the CBAs to the imperative parts of a flow. Finally, the resulting flows are validated by generating traces from them. Flows that produce deadlocks (two or more activities blocking each other due to conflicting constraints) are discarded.

Overall, we generated 165 structurally different flows and 200 traces per flow for our evaluations.

The simulation has three important independent parameters. The first one is the *navigation threshold* t_n of the flow engine as defined in Section 3.3.2. For a higher navigation threshold the flow engine accepts less uncertainty in the context events it receives. We tested t_n from 0.4 to 0.6 in steps of 0.05.



(a) Comparision of event improvement

(b) Comparision of flow completion

Figure 3.5: Simulation Results - Comparision between FlowCon and FlexCon

The second parameter is the *average recognition rate arr* of the CMS. When a context event e is created in the CMS, arr is the average probability assigned to the correct event type in the distribution I_E^e by the CMS. The remaining probability $1 - arr$ is geometrically distributed to the other event types of the respective event type set E.

The *variance v* is the third simulation parameter. It represents the noise added to the distribution I_E^e created by the CMS. The probability of each event type $u \in E$ is varied by $\pm v/2$, and I_E^e is normalized again. We evaluated the system for variance values between 0.05 and 0.6 in steps of 0.05.

To assess the performance of FlexCon we use the *relative event improvement* and the *number of completed flows* as our two metrics. The relative event improvement r is defined as $r = I_E''^e(u)/I_E^e(u)$ for the correct event type u. If $r > 1.0$, then FlexCon was able to provide additional evidence for the occurrence of the correct event type u, and the flow engine has a higher chance of making the correct navigation decision.

The number of completed flows is simply the percentage of all traces that did complete their execution successfully. We did include the learning of the model in the simulations and the execution starts without a flow history. To put our system further into perspective, we directly compare the results with our previous measurement of the same metrics in FlowCon. Note that the flows in FlowCon are purely imperative. That is, activities are connected by transitions and there are no constraints that leave the decision about the ordering of the activities to the user. Thus, the task of FlowCon is much easier than that of FlexCon due to the additional flexibility of the flows.

3.6.4.1 Results and Discussion

The evaluation results are depicted in Figure 3.8. We only show the results for $t_n = 0.4$ and $arr = 0.45$ for clarity. Furthermore, these conditions closely resemble the situation in the hospital and they can be compared best to our previous work.

Figure 3.8(a) depicts the comparison of the relative event improvement rates for FlowCon and FlexCon. The average event improvement is better for almost all variance values. Even for the higher variances of $v \geq 0.4$, where the improvement of FlowCon declines, FlexCon is able to maintain a good improvement, mainly due to the changed method of accuracy improvement: While FlowCon uses all the observed event instances as evidence for calculating the probability of the current event, FlexCon only applies the evidence for the current particle for particle propagation, i.e. independently from other particles. When we misinterpret an event instance from a preceding node this has less impact on the particle filter, as only the propagated particles from this node are influenced, but not the particles from other preceding nodes. Where in FlowCon the whole conditional probability for the current event can be distorted, in FlexCon only a partial result suffers from the misinterpretation. However, if only one parent

exists for a given node in the DBN, FlexCon is also sensitive to this kind of misinterpretation, leading to $r < 1.0$ making the result worse.

The high standard deviation for the event improvement on the flows can be explained by the flows' flexible structure. If two subsequently executed activities are not connected by a constraint or transition, we cannot improve the event in any way as there will be no connection in the DBN between the respective nodes. So according to the flow structure, we have a very high improvement for the dependent events but none for the independent ones.

Figure 3.8(b) shows the comparison of the flow completion rates, between FlowCon, FlexCon and the respective basic flow engines which do not take any action to decrease the event uncertainty. *FlowCon - Basic* and *FlexCon - Basic* simply execute the same flows without uncertainty reduction. Both basic systems fail at very low variance values. For $v \geq 0.15$ less than 6% of the flows can be completed successfully for both basic flow engines. The high values for the basic FlexCon flow engine compared to the basic FlowCon flow engine for $v = 0.05$ and $v = 0.1$ result from a changed method of generating the event instance distribution.

The FlexCon DBN-Engine manages to complete 45% of the flows at $v = 0.15$ and this performance decreases slowly for higher $v \geq 0.2$. It is still able to complete 20% of the flows at $v = 0.6$.

Again, the standard deviation on the number of completed flows is rather high, for the same reason as above. Some of the flows allow a very good event improvement leading to a reliable execution, after the training phase of the DBN is complete. Those flows (about 5% of the tested flows) exhibit an completion rate of well over 80% and are the main reason for the high standard deviation. Most of the flows are close to the average, and can complete their execution in about 30% of the cases.

3.7 FeVA – Tolerating Event Assignment Errors

In this section, we present the *Fuzzy Event Assignment* (FEvA) algorithm that enables flows to robustly interpret incoming context events and assign them to the correct activities. FEvA exploits the structural and contextual relation of flow activities and the current execution state of flows as additional information for dealing with false positive, out-of-order, and missing events. This leads to a notable improvement of the robustness of context-aware applications. On average 91% of the context events are correctly assigned, and the number of successfully completed flows increases by 52% over a reference system without FEvA.

Current CMS' provide methods for dealing with the uncertainty of primary context (e.g. location information [47]) and for composing high-level context information (e.g. recognizing tasks of a nurse for documentation). In the latter case, uncertain context reasoning or event correlation can be applied [44],[23]. However, no general method has been found to avoid arbitrary errors in context information including false positives, out-of-order events and missing events – errors that can easily happen due to timing issues, software mistakes, and especially misinterpretation. FeVA is the first system that tackles this problem.

3.7.1 Error Model

In order to execute a flow successfully, it has to be ensured that (1) the uncertainty of the events is so low, that the flow can correctly interpret them and (2) the received events are are complete and in the correct order. While FlowCon and FlexCon tackled the problem of reducing the uncertainty (see Sections 3.5 and 3.6), FeVA focuses on the difficulties that arise, when the *sequence of events* is influenced by the uncertain recognition process. We call S a *valid event sequence* (cf. Definition 3.3.4 on page 28) if it leads to a successful execution of a given flow. This successful flow execution can be disrupted by three types of errors in the event sequence that we model as follows:

The first error type are *false positives* that occur when the CMS notifies the application about an event that did not happen in the real world. We define α as the fraction of false positive events added

to a sequence S. We assume that added event instances are uniformly distributed over the sequence. Their type is randomly picked from $\mathcal{E}$, and their probability distribution is similar to that of the other event instances in S, i.e. they cannot be distinguished from correct events in S when inspecting the distribution.

The second error type are *out-of-order events*. Due to network transmission delay in a distributed CMS, temporary sensor failures or a delay in the low-level detection logic, the order of events in a valid sequence may be altered. We define γ as the fraction of events that have not been affected by a sequence shift. The affected events are shifted according to a normal distribution $\mathcal{N}(0, \sigma)$.

Finally, there are *missed events* that happened in the real world but were missed by the CMS. This might happen due to sensor unavailability or a bad reading. Let δ denote the fraction of events in a valid event sequence that have been missed. A single event sequence can be subject to all three error types and we write $S_{\alpha,\gamma,\delta}$ to express its error properties.

3.7.2 Fuzzy Event Assignment

The goal of FEvA is to interpret an incoming, erroneous event sequence $S_{\alpha',\gamma',\delta'}$ with $\alpha' > 0, \gamma' < 1, \delta' > 0$ for a flow f into the error-free original valid sequence $S_{0,1,0}$. FEvA operates inside the flow engine and monitors the flow execution closely, exploiting the knowledge about preceding and succeeding activities to detect the described errors for the current activities, flexibly correcting the event assignment.

First, we explain the so-called *activity state space* and how it is extended to catch out-of-order events correctly. Then, we describe the assignment of context events to the correct activities. An event instance e can become a *candidate* for any activity that is currently subscribed to an event type set E with $e \in E$. Of course, multiple activities may subscribe for E, and we will describe the mechanism for resolving the resulting competition for e. The *candidate selection* algorithm *fuzzifies* the incoming context events and provides them as possible candidates to the activities. An activity then decides if the event becomes a candidate and waits for further events. As the execution of a single activity progresses it will eventually have candidates for all the event type sets it registered for. Then it can complete its execution and the *event assignment* algorithm finalizes the assignment of the candidate events and resolves possible conflicts with other activities.

3.7.2.1 Flow Activity State Space

The state machine for an activity is depicted in Figure 3.6. During flow execution, an activity a can be in six different states that indicate its completion progress. These states are in their order of execution $Z = \{inactive, prepare, ready, active, can\text{-}complete, complete\}$. Let $\omega : A \rightarrow Z$ be the function that retrieves the current state of an activity a. When a flow instance is created, all activities are in the *inactive* state. An activity a that meets all prerequisites for being executed switches to the *ready* state. The flow engine then registers a's event type sets at the CMS. After a has been informed of the arrival of the first event instance with an appropriate event type set, a reaches the *active* state. When a has been informed that for all registered event type sets, event instances have arrived, the conditions of the outgoing transitions $(a, a_x) \in T$ are evaluated and a reaches the *complete* state. The target activities a_x of the outgoing transitions, whose conditions have been evaluated to true, are set to the *ready* state. The execution of the whole flow instance is considered successful if no activity is currently running (i.e. in the *active* state) and all activities that are mandatory for the successful execution have reached the *complete* state. The *prepare* and *can-complete* states have been added during the development of FEvA.

First, if the state $\omega(a_y)$ of each preceding activity a_y with $(a_y, a) \in T$ is $\notin \{inactive, prepare\}$, then a switches from the *inactive* to the *prepare* state. The event type sets of an activity in the *prepare* state are also registered at the CMS. Therefore, the risk of missing an out-of-order event that arrives earlier than expected are reduced, because the activities also register early for their event type sets.

Second, before switching from *active* to *complete*, an activity a reaches the *can-complete* state first. This state indicates that a has found candidates for all its event type sets but the preceding activities

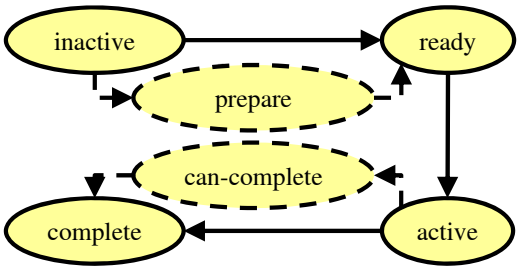


Figure 3.6: Activity State Machine

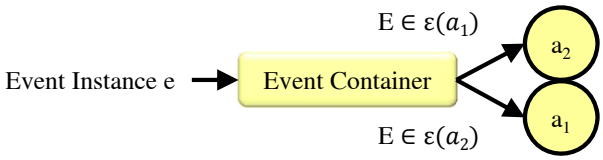


Figure 3.7: Event Container Principle

have not yet reached the *complete* state. This can happen when events have been missed or arrived out-of-order. Waiting for the completion of the preceding activities, we avoid that a consumes events that are possibly more suitable candidates for the predecessors while a better event for a might still arrive. However, the conflict resolution mechanisms, which we will introduce along with the event assignment algorithm, will occasionally bypass this rule to handle missed events (cf. Section 3.7.2.2). For more details on the formal flow model, the execution semantics and condition evaluation we refer to our previous work [79].

3.7.2.2 Candidate Selection and Assignment

FEvA consists of two algorithms, one for event candidate selection and one for event assignment. Both algorithms are plugged into the *event container* (cf. Figure 3.7), the component of the flow engine responsible for event caching and dispatching. The event container is notified whenever an activity a registers its event type sets $\epsilon(a)$ at the CMS. The event container also updates the the *set of competing activities* $C_E = \{a \in A \mid (\omega(a) \notin \{inactive, complete\} \wedge E \in \epsilon(a)\}$ for each event of type set E. Furthermore, the event container stores a list of *candidate events* for each a and E denoted as $candidates(a, E)$. All new event instances, the flow engine is notified about, are cached in the event container.

3.7.2.3 Candidate Selection

The candidate selection algorithm, depicted in algorithm 7, computes which event instances are added to the list of candidates of an activity. Since we only get the probability distribution I_E^e for every event e, there is no hard criterion for deciding which type e has, but only probabilities. Therefore, we use fuzzy set theory (cf. [81]) to value every event, assign them to event types, and finally to activities. First, the algorithm computes a fuzzified representation of e. We use fuzzy sets, each representing a linguistic value, defining the fitting quality of e for a single event type $u \in E$. The individual fuzzy membership functions are defined as $\mu_x : [0, 1] \rightarrow [0, 1]$ where $x \in \{VL, L, M, H, VH\}$ is one of the linguistic values "very low", "low", "medium", "high", "very high". Each function μ_x maps the probability $I_E^e(u)$ for e being of a single event type u to a fuzzy membership value for the respective linguistic value. We use the same membership functions μ_x based on the standard triangular fuzzy functions [59] for all combinations of activities and event type sets. For example, $u \in E$ is the event type representing that the nurse has measured the pulse of the patient and $I_E^e(u) = 0.375$ then $\mu_M(I_E^e(u)) = 0.75$ and $\mu_H(I_E^e(u)) = 0.25$.

As each event is weighted by every membership function, we further introduce the *fuzzy event type weighting* function $\lambda : [0,1] \rightarrow [0,1]^5$ as a concise version including all the membership function results. Given $u \in E$, $\lambda(I_E^e(u))$ yields $(\mu_{VL}(I_E^e(u)), \mu_L(I_E^e(u)), \mu_M(I_E^e(u)), \mu_H(I_E^e(u)), \mu_{VH}(I_E^e(u)))$, i.e. the mapping of the individual probability of the event type to the fuzzified membership in all five fuzzy sets. The maximum membership values for each variable are given as follows: $\mu_{VL}(0.15) = 1.0$, $\mu_L(0.25) = 1.0$, $\mu_M(0.35) = 1.0$, $\mu_H(0.45) = 1.0$, $\mu_{VH}(0.55\ ) = 1.0$. For $p < 0.15$, $\mu_{VL} = 1.0$ and for $p > 0.55$, $\mu_{VH} = 1.0$.

For all $u \in E$, the candidate selection algorithm weights u with the *fuzzy event type weighting* function $\lambda(I_E^e(u))$ and notifies the activities in C_E, i.e those that have subscribed for E, about the result. Using the *fuzzy activity weighting* function $\kappa_a : [0,1]^5 \rightarrow [true, false]$ of the activity a, the decision is made whether e becomes a candidate for a or not. κ is derived from the structure of the conditions of a applying fuzzy logic. It might be modified per activity by the flow modeler, but we only use the automatically defined κ for a.

To explain how the result of κ_a is computed, let $u \in E$ denote the pulse measuring event type again. Further, a has a condition that requires $u \in E$ to have happened, so that the respective transition can evaluate to *true*. The result of κ_a becomes true if and only if the lowest non-zero linguistic membership in $\lambda(I_E^e(u))$ is "high" or "very high". Thus, $\mu_H((I_E^e(u)) \geq 0.0 \vee \mu_{VH}(I_E^e(u)) \geq 0.0)$. This also represents the minimum *candidate threshold* for an event to be accepted as a candidate for any activity. Given this equation is fulfilled, the result of $\kappa_a(\lambda(I_E^e(u)))$ yields *true* and e is stored as a possible candidate for the activity a and the event type set E in $candidate(a, E)$.

If e becomes a candidate for an activity a, FEvA further checks if e has the best overall fitting of the candidates available in $candidates(a, E)$. Let $u \in E$ be the event type that a is interested in. We denote the event instance with the highest fitting as e^{max} where $\forall e \in candidates(a, E) : \mu_x(I_E^{e^{max}}(u)) \geq \mu_x(I_E^e(u))$ for the highest non-zero linguistic membership value of e^{max}. Given that the new event instance e is the new best fitting one ($e = e^{max}$), the algorithm issues an assignment request for e that is later handled by the event assignment algorithm.

Algorithm 7 Candidate Selection Algorithm

```
Input: C_E, e
for u ∈ E do
    fuzzyWeights(u) ← λ(I_E^e(u))
end for
for a ∈ C_E do
    for u ∈ E do
        if κ_a(fuzzyWeights(u)) then
            candidates(a, E) ← candidates(a, E) ∪ {e}
        end if
    end for
    e^max ← max(candidates(a, E))
    issue_assignment_request(a, e^max)
end for
```

3.7.2.4 Event Assignment

When an activity a is eventually notified that for all registered event type sets suitable candidates have been found, its state changes to *can-complete*. Now the activity must consume a single candidate for each event type set from the event container, before it can commit its execution and reach the final *complete* state. The event assignment algorithm is responsible for processing the issued event assignment requests of the activity. However, in the assignment of an event instance e to a, conflicts might occur

because it could also have been requested by another activity a_o. The event assignment algorithm[2] is also responsible for resolving such conflicts:

If there is an $a_o \in C_E$ that has also issued an assignment request for $e \in E$, we search for an alternative candidate in $candidates(a_o, E)$. If this alternative is available, i.e $(candidates(a_o, E) \setminus e) \neq \emptyset$, then a consumes e and the algorithm computes e^{max} again for a_o and its changed set of candidates. When there is no other candidate available in $candidates(a_o, E)$, we check if only one of the activities is mandatory, and prefer to assign the event to the mandatory activity. If a cannot assign an event using these two mechanisms, then we try to re-evaluate candidates that have been rejected earlier. In order to do so, we relax the candidate threshold for the missing event, if at least one of the other event type sets have a very good candidate assigned. This is the case if there is an assignment request for one event type set of $\epsilon(a) \setminus E$, that exceeds the candidate threshold, having a nonzero "very high" fuzzy weight, or more formally for $E' \in (\epsilon(a) \setminus E) : \exists u \in E', \exists e' \in candidates(a, E') : \mu_{VH}(I_E^{e'}(u)) \geq 0.0$. Given this equation is fulfilled, then the threshold for the missing event type set E is reduced to "medium" and events, which are still cached, could now become also candidates for a.

If this also does not yield a suitable candidate, the activity may be completed without having been assigned an event for E, under the assumption that the event instance for E was missed. But in order to do this, a number of strict criteria have to be fulfilled. First, there is a fixed maximum number of allowed missing events per activity. As the average number of event type sets per activity in our flows is ≈ 3, we accept only one missing event, so that the number of missing events is always below or equal to the number of received events per activity.

Second, the preceding activities of a must be *complete* and the succeeding activities must be in *can-complete*, because this indicates that all other events before and after the activity in questions have been identified or already assigned. Third, the succeeding activities must have at least one candidate with a very high fitting value. These rules contribute enough evidence to decide, that the activity can be completed without the missing event. In this case, we violate the transition rule for the *can-complete* state given in Section 3.7.2.1.

3.7.3 Evaluation

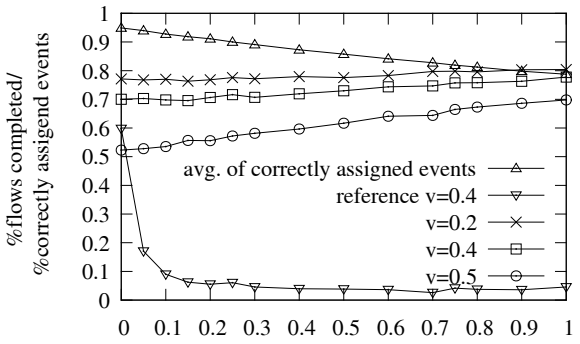
To evaluate FEvA with respect to the error model we introduced in Section 3.7.1, we extended our existing simulation environment, developed in previous work[79]. In the following, we present the setup of the individual experiments and the simulation parameters and then discuss the results.

3.7.4 Simulation Setup

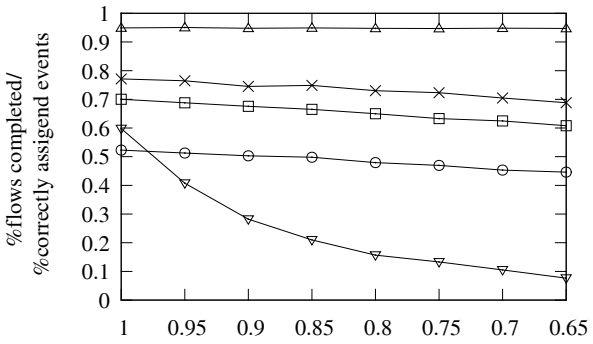
We need a suitable set of realistic flow models to evaluate FEvA in a wide range of cases. However, there is no sufficiently large set of different real-world flow models publicly available for us to use in this work. Therefore, we created the flows from so-called *workflow patterns*. These patterns are common building blocks that have been identified in a number of real-world workflows, which include a high number of human tasks [22]. Furthermore, it has been shown that these patterns adhere to a co-occurrence distribution, indicating that some patterns follow others more regularly [48]. We used this co-occurrence distribution to generate the structure of the workflows used in our evaluations. The flows had sizes of 20, 30, 40, and 50 activities. For each size, we simulated 25 different flows and fed 100 different event sequences into each flow. Therefore, every data point in the evaluation results is created from 10,000 flows executions.

The simulation is set up using two sets of parameters. The first set consists of the two values: the *ground truth GT* and the *variance V*, which are used to generate I_E^e. GT is the probability that the CMS detects the correct situation that happened in the real world. The remaining probability $1 - GT$ is geometrically distributed to the other events types of the event type set. We have estimated the lower

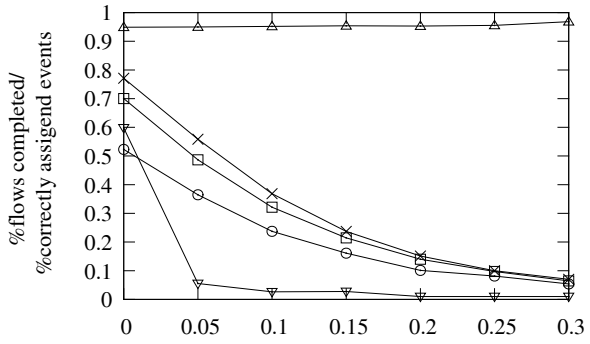
[2]We omit a listing of the algorithm due to space restrictions.



(a) fraction of added events α



(b) fraction of in-order events γ



(c) fraction of deleted events δ

Figure 3.8: Simulation Results - FEvA performance: The graphs for "reference v=0.4", "v=0.2", "v=0.4", and "v=0.5" show the percentage of flows completed. The graphs "reference v=0.4" depict the result of running the flows without FEvA

range of $GT = 0.4$ based on real-world context recognition experiments conducted in a real hospital scenario [5]. We varied GT between the measured 0.4 and 1.0 in steps of 0.1, where $GT = 1.0$ yields always the correct recognition. V adds noise to the resulting distribution, i.e. the probability of each event type is altered by V, and the final PDF is normalized again. We chose V between 0.05 and 1.0 in steps of 0.05. To enhance readability, the figures only show the graphs for $GT = 0.5$ which is FEvA's threshold for accepting an event (cf. Section 3.7.2.2) and variances of $0.2, 0.4, 0.5$ representing a low, medium high, and high amount of noise. The second parameter set consists of α, γ and δ according to our error model in Section 3.7.1. For α we chose values between 0.0 and 1.0, for γ between 1.0 and 0.65 and for δ between 0.0 and 0.3. The weighting functions remain as introduced in Section 3.7.2.2

3.7.5 Results

We evaluated each of the parameters independently and the results of the simulation for each parameter α, γ, and δ are depicted in Figure 3.8. On the y-axis of each graph, we denote the fraction of flows that could successfully complete their execution and also the fraction of total events that have been assigned to the correct activity during the execution. Each graph also shows the performance of our system for $V = 0.4$ without using FEvA, but a naive assignment approach as reference.

The results for the added false positives are the most surprising (cf. Figure 3.8(a)). While the number of correct events assigned to the activities decreases slowly from about 94% to 78%, the actual number of successfully completed flows increases. This is due to the fact, that the added false positives also take part in the assignment process and, especially in conflict situations, may be mapped to an activity

too early and falsely. This effect becomes stronger the higher the variance. If V is lower and events are recognized accurately, the flow is able to deal with more false positives, without a strong impact on performance. However, the higher the variance the more counter-intuitive the result is and the more flows complete because they have assigned false positives. Compared to the reference without FEvA we perform at least 10% better and the other system basically fails even at low rates of false positives.

When we confront FEvA with out-of-order events (cf. Figure 3.8(b)), the algorithm performs very well and tolerates the deviation. Most of the events are correctly assigned, i.e. well over 97% and the impact of the few falsely assigned events on the flow completion is low compared to the variance. Again compared to the reference we are up to 52% better.

Considering the missed events (cf. Figure 3.8(c)), FEvA is still able to assign the remaining events accurately, again with well over 97%, but the missing events have a very strong impact on the flow execution. While a low number of missing events is somewhat tolerable, the amount of correctly completed flows drops rapidly to a mere 7% when more than a quarter of all events is missing. The mechanism we introduced to tolerate missed events performs better in up to 40% of all cases compared to the reference. However, there is still a lot of room for improvement, e.g. by also using information from the assigned events.

3.8 Summary and Conclusions

In this chapter, we have presented three systems that we designed in the project for the purpose of making flow navigation more robust. FlowCon and FlexCon are novel approaches that increases the accuracy of context information in flow-based pervasive applications. Both use Bayesian Network technology to learn the correlation between context events. Using this learned knowledge, both systems provide additional evidence for the occurrence of events, and this facilitates the threshold-based navigation decisions in a flow engine.

While FlowCon handles classical imperative flows, FlexCon is able to deal with the more flexible and user-friendly hybrid flows. Both show big improvements in flow robustness over a system that does not handle context information uncertainty. Based on real-world traces gathered in a geriatric nursing home, we have shown that FlowCon achieves a significant decrease of event uncertainty of up to 49%. Furthermore, it makes the execution of flow-based mobile applications more robust. The ratio of flows that could complete their execution successfully was increased significantly to up to 90%. Our evaluations for FlexCon show that the uncertainty of an event received by a flow is reduced by 54% on average and the percentage of successfully completed flows is increased by 23-40%.

Furthermore, we have presented FEvA, a system that exploits the knowledge encoded in a workflow in order to assign noisy and erroneous incoming context events to the correct activities in the workflow. We have demonstrated the effectiveness of FEvA under a well-defined error model and showed that it achieves a high robustness when faced with a large number of false positive events. It also works very well in the presence of out-of-order events, and it limits the impact of missing events.

This work is an important step towards applying flow technology as a part of pervasive systems. In real-world scenarios, found e.g. in the health care domain, users need to be supported in their activities without obstructing them. Thus, the flows need to automatically synchronize with their activities based on collected data such that users are not required to communicate with the flow explicitly. Especially in the health care domain, any explicit interaction (using touch screens etc.) may have severe implications in terms of hygiene.

The sensor data that is used to infer the current activity of a user is characterized by a high level of noise and inaccuracy. FlowCon and FlexCon offer a way to infer more reliable information from this data and, thus, render the respective flows more robust.

Chapter 4

Context Prediction

Even though the initial vision of pervasive computing already dates back over a decade ago, the demand for intelligent and user-centric technology is still a major challenge in the focus of current research. Key for the development of pervasive technology which operates unnoticed from, but on behalf of the user, is seen in the principle of context awareness. While advances in recognition and processing of context have spawned a variety of new context-aware applications, these applications are often characterized by *reactive behavior* and respond to changes in the current context *after* their actual occurrence in the real world.

As a consequence, the level of intelligence found in these environments is restricted to what the user already can observe in his current situation. In order to render these environments more intelligent, applications should anticipate the future needs of users even *before* they physically appear [58]. This enables the provision of proactive pervasive services that improve the experience of a user when interacting with his environment.

Examples can be found in many relevant fields of pervasive computing such as for instance home automation, information recommendation services as well as human guidance in working environments. As the human needs in pervasive scenarios are tightly coupled to their future context, sophisticated methods for *context prediction* are required to extend the temporal horizon of context awareness into the future. The most common approach for context prediction is to rely on context histories to deduce probable future behaviour from the sequences of past context changes [52]. However, without any insight into the application domain, the history analysis might result in ambiguous context predictions. The reason is that the context transitions recorded in the histories are decoupled from the semantics of human behaviour. The prediction may then result from history patterns that are not relevant to the current user situation. In contrast, many application domains of pervasive computing such as pervasive healthcare are characterized by human models of structured behaviour. Higher-level knowledge about the activities of a human provides the possibility to filter context histories more accurately.

In this chapter, we show how to exploit flows as source of information to provide history-based predictors with domain knowledge for accurate context predictions. For this purpose, we first introduce a novel flow-based predictor in Section 4.2 that relates context changes to states in human behaviour. Our predictor is based on a probabilistic state transition system that is learnt from the execution of flows and defines the search space for future context occurrences. For a prediction, we determine the most likely sequence of future context states reachable from the current user situation. As these search paths depend on the activities performed by humans, we are able to derive accurate context predictions that remain hidden for history-only approaches.

In Section 4.3, we present PreCon, an approach to context prediction that allows time-dependent multi-dimensional context prediction queries. PreCon is based on the system introduced in Section 4.2 and applies well-known methods of *stochastic model checking* [9] (used e.g. for the verification of distributed communication protocols) to the analysis and prediction of human behaviour. While classical model checking relies on fixed hand-crafted models of computer systems, our models (called *stochastic*

user models (SUM) in the following) are representations of human behaviour that are learned from traces of past context changes. We represent SUMs as *Semi-Markov Chains* such that the changes in context are regarded as a stochastic process. We use *temporal logics* as a query language, enabling applications to specify expressive temporal properties on future context. For a prediction, our system verifies with which probability these properties hold on a given SUMs. The result is a comprehensive framework to provide accurate knowledge about future context changes to flow-based systems in order to trigger proactive actions and support processes of efficient context-aware decision-making.

4.1 Related Work

The most comprehensive approach to context prediction has been presented by Mayrhofer [53], who proposes a multi-layer system architecture for domain-independent context recognition and prediction. This work targets the prediction of high-level user context (context classes such as "in a meeting") which is derived from low-level context data (e.g. location, noise, etc.) in a preceding classification step. In order to improve the achieved prediction results, the author concludes that the inclusion of domain-specific knowledge would represent a promising approach, as addressed by our work. Sigg [68] directly predicts low-level context before deriving future high-level context in order to avoid information loss resulting from the aggregation of low-level context. However, this approach does not consider possible correlations among low-level context and deduces future context based on matched sequences from the past without taking advantage of domain-specific knowledge about human behaviour.

Furthermore, different algorithms have been proposed to allow for the prediction of specific categories of context. Most of the work in this area focuses on the prediction of user mobility from location histories [14], [70] [43], [6]. Beyond a history of location sequences, these approaches do not consider any further information for prediction. However, location predictors for wireless cellular networks can exploit the structure of geographic areas and direction information to anticipate future location changes more accurately [21]. In contrast to these approaches, we argue that the behaviour of humans is the most valuable information for prediction. Moreover, the prediction of low-level user activities such as key pressings to improve the interaction with user interfaces has been studied [36], [25], [31]. Similar to location prediction, future user activities are derived from histories of past sequences. In our approach, activities are the constituent parts of context-aware workflows that synchronize with the behaviour of humans in the real world. The difference is that we use the knowledge provided by the workflows to predict additional context that evolves with the activities performed by users. The idea to take advantage of domain knowledge in order to improve the recognition of hidden patterns from data is inherent to the field of syntactic pattern recognition [66]. Based on a statistical model that describes the generation of the data, patterns such as handwriting symbols or actions [17] can be discovered more accurately. Following this line of argumentation, we argue that also context prediction can benefit from a structural model of human behaviour that can be found in relevant fields of pervasive computing such as pervasive healthcare.

Current approaches to context prediction [70], [6], [43], [36], [31], [25] only allow for very simple queries for the most probable next user context (e.g. the user's next location). Hence, their expressiveness is severely limited since applications are not able to extract any information on the expected time of such a context change. We argue that intelligent ubiquitous applications need therefore the ability to submit *time-dependent queries* (e.g. "Will user *A* be at location *x* within the next 10 minutes?"). Likewise, queries for multi-dimensional context (e.g. "Will the user be executing *activity Y* at *location X* within the next 10 minutes?") must be possible. PreCon can answer such expressive queries accurately, and thus, provides applications with more flexibility and allows them to act in a more goal-oriented way for their user. However, they also did not consider a sophisticated query language that supports temporal relations. PreCon is the first system to investigate the application of temporal logics as a powerful and expressive query language for context predictions.

For the calculation of the predictions, we adopt the techniques from the field of stochastic model

checking [46], which has been thoroughly studied by Baier et al. for Continuous Time Markov Chains [8]. Lopez et. al have worked on the extension of the model checking algorithms for Semi Markov Chains [51]. In our approach, we leverage on their results. However, we extend their original model checking algorithms by including the running dwell time of the states, allowing for accurate real-time predictions.Semi-Markov Chains as we use them have also been employed by Lee and Hou for an analysis of transient and stationary features of user mobility on the Dartmouth campus network [49]. However, the predictions supported by these approaches only compute the most probable next context in a single-dimensional context space. They do not allow queries for temporal relations in a multi-dimensional context space. In contrast to classical model checking approaches, PreCon does not rely on a design-time specification of a state transition system. Instead, we apply a learning algorithm for building an SMC in an online fashion. This allows us to incorporate newly available data at any time, making the predictions more accurate. The combination of SMCs, a query language based on temporal logic, and the online learning approach represents an important new step in the area of context prediction.

4.2 Flow-based Context Prediction

4.2.1 History-based Prediction

In the following section, we focus on history-based methods for context prediction. First, we will present the common procedure of history-based predictors. Then, we will analyse their shortcomings for prediction in flow-oriented environments.

4.2.1.1 Context Prediction

The rationale of context prediction is to extract characteristic patterns in human behaviour from histories of observed context data. As human behaviour cannot be captured exactly, the most common approach is to apply stochastic principles to describe the expected changes in user context. For this purpose, the occurrence of context (e.g. location) is regarded as a random variable X, which can be assigned values from a discrete set of context elements $C = \{id_1, id_2, .., id_n\}$. We assume that each context $c \in C$ can be associated with a unique symbolic identifier. For example, in terms of geographic positioning, symbolic location names such as "office" or "kitchen" provide a meaningful attribute of the user's location. Let the context history $H = c_1, c_2, ..., c_n$ be defined as a sequence of context elements $c_i \in C$ ordered according to their time of occurrence. Sequential changes in context can thus be considered as a stochastic process χ that describes the evolution in user behaviour with distribution $P(X_1 = c_1, X_2 = c_2, ..., X_n = c_n)$.

The most widely employed history-based predictors from related work [14, 31, 70] are based on discrete Markov processes. The Markov assumption is inherent to two different classes of predictors - the fixed order *0(k)* Markov predictors and the predictors based on varying order Markov models. $0(k)$ Markov predictors consider a fixed window of past context observations for prediction. The order k of the Markov predictor determines the length of the window that influences the predicted context. Consequently, the part of the history relevant for prediction is given by the k last observations $H(k) = c_{n-k+1}, ..., c_n$. Assuming a stationary stochastic process, the conditional probability distribution can be estimated from the occurrence of context changes in the history. The prediction is then determined by the context c_{n+1} which has most frequently followed the sub-sequence $H(k)$ in the entire history H. The restriction of a fixed order is relaxed by so-called Markov Models of varying orders. The most popular varying order Markov predictors are based on the data compression algorithm of Ziv and Lempel [82].

In order to incorporate domain knowledge, our prediction scheme is based on a generic model of history-based predictors. We extend this generic model in Section 4.2.3 to combine it with knowledge about user activities. A very important consequence of this technique is that it allows us to apply our approach to several variants of state-of-the-art context predictors. In our generic model, we define a history-based predictor as a probabilistic state transitions system. The definition captures the common

nature of Markov predictors: A state is a sequence of one or more past context elements upon which predictions are derived from the analysis of context histories.

Definition 1 (History-based context predictor): A history-based context predictor $\hat{P}$ is specified by a probabilistic state transition system (S, C, δ, p), where

- C is the set of discrete context elements
- S denotes the set of history states
- $\delta : S \times C \rightarrow S$ denotes the transition function that describes possible context changes
- $p : S \times C \rightarrow [0, 1]$ indicates the probability for a specific context change

Each $s \in S$ corresponds to a history state. When observing a context $c \in C$ during state s, the history state changes to $s' = \delta(s, c)$. Thus, the history state evolves with new context observations. However, the transition function is partial as not necessarily each context can be observed during a history state. The predictor encodes the probability of future context occurrences in transition probabilities. The probability $P(c|s)$ to expect a context $c \in C$ depends on the history state s and is indicated by $p(s, c)$. The sum of all probabilities over all outgoing transition has to be one, i.e., $\forall s \in S : \sum_{\forall c \in C; \delta(s,c) \neq \emptyset} p(s, c) = 1$. Initial states, as known from classical automaton theory, are defined by the current history state at time of prediction.

4.2.1.2 Analysis of History Restrictions

The limiting factor of history-based predictors is implied by their nature – their dependency on past observations as the only indicator to what can follow next. Due to the Markov assumption, the sequence of past context occurrences must carry enough information to make accurate predictions. However, historical information may not be sufficient for enabling accurate predictions in every case. This observation is especially relevant in cases where the situation of a user is defined in terms of higher-level behaviour that more precisely implies the next context to occur. The accuracy of predictions is low if the next context has a semantic association with the user's behaviour that cannot be learnt from the history. If we imagine for example a worker leaving his office, then there are many equally likely options for his future location according to the information from the location history. If we could access the knowledge that the worker decided to visit a customer, we could use the domain insight to forecast his next location more accurately. Here, the capabilities of a history predictor are very limited, because the knowledge from the history is not able to capture this form of hidden information.

Taking the user behaviour into account, we are able to provide history predictors with the necessary knowledge to reduce ambiguities. As a consequence, we require a model of human behaviour that allows us to interpret situations, based on a human's past and future activities and the context under which the activities are taking place. For this purpose, we will leverage on *Adaptable Pervasive Flows* as a context-aware model of human behaviour as described in the next section.

4.2.2 Exploiting Flow-Knowledge

In this work, we exploit flows as providers of domain-specific knowledge for context prediction. The knowledge stems from the fact that a) a flow provides insight in the current state of its associated entity, and b) a flow models paths of future activities. This enables us to develop a generic flow-based context prediction scheme.

For the purpose of context prediction, we do not rely on a specific technology and focus on the generic model of flows. A *flow model* describes the activities of a human under changing contextual conditions. As an example consider the flow attached to a nurse in a hospital shown in Figure 4.1. During her work day, a nurse carries out regular activities such as "give medication" or "serve lunch". After the

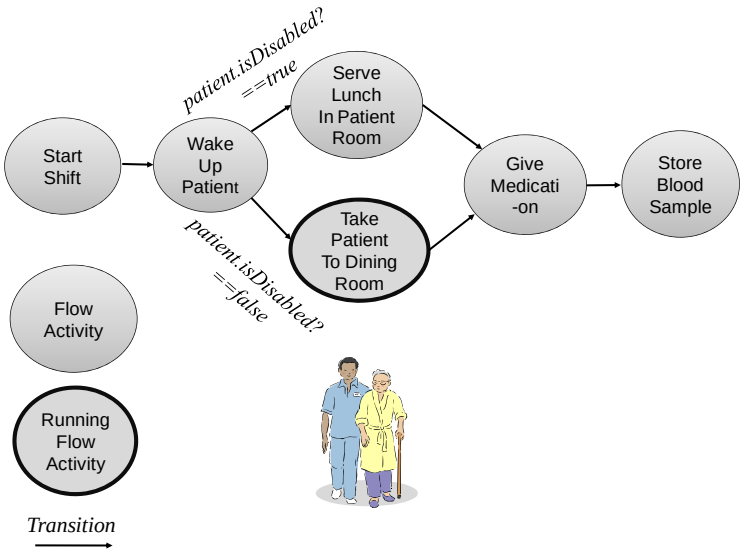


Figure 4.1: Representation of a Pervasive Flow attached to a Nurse

start of her shift she progresses in her workflow and executes activities that are associated with contextual conditions. For example, the concrete activities of a nurse depend on the health conditions of the patient she is caring for. For the specific purposes of this chapter, we define what a flow model is. This definition is similar to Definition 3.3.5 on page 28 but differs in some details that are relevant for the design of our prediction scheme.

Definition 2 (Flow Model): A Flow Model f is specified by a directed graph $(A, E, P(C), t)$, where

- A denotes the set of *activities*
- $E \subseteq A \times A$ defines a *control flow*
- $P(C)$ is the set of predicates over the *entity context* C
- $t : E \rightarrow P(C)$ associates each control link with a transition condition

The flow model defines a *control flow* over the set of flow *activities* based on a directed graph. Each flow contains a start activity from which there is a path to any other activity in the flow. The control flow constraints the possible paths of activity executions. An activity path $a_1 \rightarrow a_2, ..., a_{n-1} \rightarrow a_n$ in the flow consists of pairs of connected activities, i.e., $(a_i, a_{i+1}) \in E$. An activity can only be executed, if one of its predecessor has been completed. The completion of activities is triggered by conditions that are checked at run-time. These conditions are related to the context of a human (also more generally called *entity*) in its current situation. For this purpose, context recognition techniques such activity sensing or other sources of information (e.g. patient data) are used. Through the integration of context information, the flow is synchronized with the real-world behaviour of humans.

The run-time representation of a flow is referred to as *flow instance*. Flow instances are created in a context-aware manner based on contextual triggers. For example, as soon as a nurse starts her shift, a flow instance is created and attached to her. A flow instance exposes the state of a flow, which dynamically evolves during its lifetime.

Definition 3 (Flow State): The state of a flow f is specified by its currently running activity, which is given by the function $state : f \mapsto a \in A$.

The state of a flow instance is controlled by a *flow engine*. A flow engine runs flows and accesses

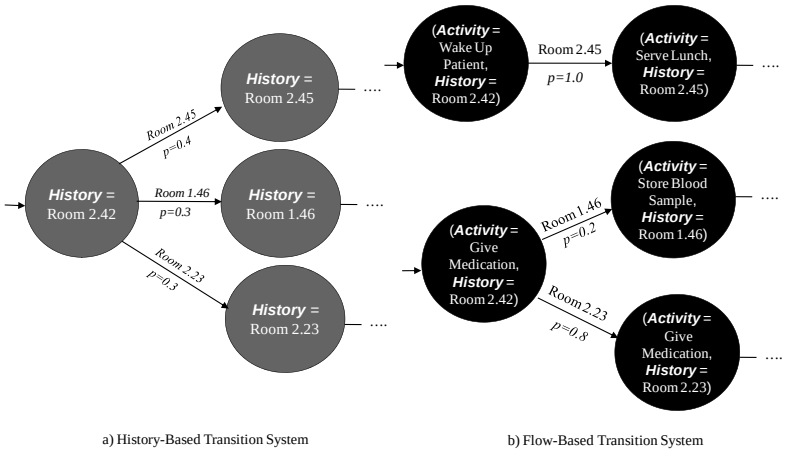


Figure 4.2: History- vs. Flow-based Transition Systems

the context information that influences their states. The state of a flow is always well defined, i.e., there is no state in between two activities. If a successor activity starts, the preceding activity is terminated. Thus, a flow state is active for a certain period of time.

4.2.3 A Flow-Based Predictor

In this section, we introduce a new *Flow Predictor* that combines both sources of knowledge – context histories *and* flows – to leverage the additional information present in flows. The relationship of flows and context is encoded as a probabilistic state transition system that includes activity information. To express this relationship, we need to extend the history-based model for context prediction.

Definition 4 (Flow Predictor): A flow predictor $\hat{P}_f$ is associated with a flow f and is based on a history predictor $\hat{P}$. $\hat{P}_f$ is formally defined as probabilistic state transition system $(\hat{S}, C, \tau, p, f)$, where

- the states $\hat{S} = (S \times A)$ are the Cartesian product of the states of the history-based predictor $\hat{P}$ and activities of flow f
- C is the set of discrete context elements
- $\tau \subseteq \hat{S} \times C \times \hat{S}$ denotes the transition relation
- $p : \tau \rightarrow [0, 1]$ indicates the transition probability with $\forall s \in \hat{S} : \sum_{\forall c \in C, s' \in \hat{S} : (s,c,s') \in \tau} p(s, c, s') = 1$

States are now defined as tuples of flow activities and history states. Thus, we establish a relation among both. The flow activities in the flow predictor introduce a new differentiating criterion for history states. We can now find the *same* history state in different states of the flow predictor. Each of these history states may be associated with distinct transition probabilities. This enables accurate predictions tailored to the current user activity. In contrast, a history-based predictor represents each history state only once.

Figure 4.2 illustrates this difference for the prediction of the nurse's location. The history-based predictor (left side) contains three possibilities for changing from "*Room*2.42" to other locations. All of these locations are almost equally probable. In contrast to this, the flow predictor (right side) links locations to activities. Note that the state relating to "*Room*2.42" has been split in two states, each being associated with a different activity ("*GiveMedication*") and ("*WakeUpPatient*"). Using this activity information, the flow predictor can make a much more reliable prediction: When the nurse is executing the activity "*GiveMedication*" in "*Room*2.42", only two out of the three locations are likely to be visited,

Algorithm 8 Online Learning

Require: Flow f

$\tau \leftarrow \{\}, buffer \leftarrow \{\}$
$h \leftarrow \epsilon, a_n \leftarrow state(f)$
while true **do**
  $e \leftarrow nextElementFromHistory(H_f)$
  if $e \in A$ **then**
    $buffer \leftarrow buffer \cup \{a_n\}$
    $a_n \leftarrow e$
    if $h \neq \epsilon$ **then**
      $\hat{S} = \hat{S} \cup \{(h, a_n)\}$
    end if
  end if
  if $e \in C$ **then**
    $h' \leftarrow \delta(h, e)$
    if $h \neq \epsilon$ **then**
      $\hat{S} = \hat{S} \cup \{(h', a_n)\}$
      $\tau \leftarrow \tau \cup \{((h, a_n), e, (h', a_n))\}$
      increment $count((h, a_n), e, (h', a_n))$
      for all $a_{past} \in buffer$ **do**
        $\tau \leftarrow \tau \cup \{((h, a_{past}), e, (h', a_n))\}$
        increment $count((h, a_{past}), e, (h', a_n))$
      end for
    end if
    $h \leftarrow h'$
    $buffer \leftarrow \{\}$
  end if
end while

and the highest probability associated with an outgoing transition has increased over the history-based predictor. Moreover, if the nurse is executing "*WakeUpPatient*" in the same location "*Room*2.42", then only one possibility remains for the next location ("*Room*2.45"). Thus, a combination with activity information extracted from flows splits a single history state into multiple states such that for each of the following states the probability may increase.

4.2.3.1 Online Learning from Flow-Enhanced Context Histories

The flow predictor provides the formal framework for incorporating user activity information into context predictions. However, the transition system underlying the flow predictor has to be learnt from the execution history of flows. The goal of learning is to obtain the state transitions and associated probabilities that accurately describe the evolution of the entity's context in the target domain.

The information used for learning is a sequence of changes in either the flow state or the context. This information is stored in the so-called *flow-enhanced history*, denoted by H_f. The changes in flow activities and context appearing in the real world (and thus also in H_f) can be arbitrarily interleaved. Thus, the flow-enhanced history is defined as $H_f = c_0, a_1, c_1, a_2, ..., c_n$ with $c_i \in C \cup \{\epsilon\}, a_j \in A \cup \{\epsilon\}$ where ϵ denotes the empty symbol. H_f is a sequential stream of events to which each new observation is appended at run-time. This is necessary, since our predictor is run in an on-line manner, i.e., the training phase is executed simultaneously to the execution of the flow-based system at runtime. The probability for future context occurrences are estimated from the observed frequencies in H_f. For this purpose, we associate a

counter with each transition $t = ((h, a), c, (h', a')) \in \tau$, denoted as $count(t)$, which stores the frequency of past transitions.

Algorithm 8 shows the procedure executed for online learning: Each new element of H_f can be either a new activity or a new context. We insert new transitions in the predictor only if a context change happens. If the last added state is (h, a_n) and a new context c is observed, we insert a transition $((h, a_n), c, (h', a_n))$ (line 16). The representation of h and h' depends on the underlying Markov model. For example, in case of a $0(1)$ Markov model, the new history state is defined only by the observed context c, i.e., $h' = c$. In contrast, for higher order or varying order Markov models further past context observations contribute to the new history state h'. Since an arbitrary series of activity changes may happen in H_f before the next context change to c occurs, we buffer these activities (line 6). Once, the new context c occurs, we also add a transition $((h, a_{past}), c, (h', a_n))$ for each of these buffered past activities (lines 18-21) since for each state (h, a_{past}) the next context is c. Suppose, for example, that the current predictor state is $s_1 = (history = "Room2.42", activity = "GiveMedication")$ and the next elements in H_f are "StoreBloodSample" (activity change) and "Room2.43" (context change). Then s_1 *and* the state $s_2 = (history = "Room2.42", activity = "StoreBloodSample")$ should be related to the next context and we insert a transition to state $s_3 = (history = "Room2.43", activity = "StoreBloodSample")$ from both s_1 and s_2. After this step, the buffer is emptied and we wait for next observed event.

The probability of a transition $t = ((h, a), c, (h', a')) \in \tau$ can be derived as a relative frequency measure from the transition counters:

$$p((h, a), c, (h', a')) = \frac{count((h,a),c,(h',a'))}{\sum_{c' \in C} \sum_{a'' \in A} count((h,a),c',(\delta(h,c'),a''))}$$

For the calculation of $p((h, a), c, (h', a'))$, we take all outgoing transitions from the state (h, a) into account. The probability is derived from the transition counters whenever a prediction has to be made. Due to the representation of activity information, there is an increased cost in storage associated with the flow predictor. The state space is now of size $O(|S| \cdot |A|)$, since activities are combined with the context from histories. Consequently, also the encoding of transitions requires more space and has complexity $O(|S| \cdot |C| \cdot |A|^2)$. In the scenarios addressed by this work, the cost will be affordable, as we assume a limited set of activities to be of interest and the real cost to be significantly below the worst case estimation, as activities are not observable at each context.

4.2.3.2 Calculation of Predictions

For predicting future context we distinguish between two classes of prediction - *short-term* and *long-term* prediction. In both cases we are interested in future context occurrences, that follow the current context history H. Let c_n denote the last context observed from H. For short-term prediction the goal is to determine the context c_{n+1} that will most probably occur next. Long-term prediction extends the time horizon to more distant points in the future. For this purpose, we define the number of future occurrences as *prediction horizon*. Formally, for a prediction horizon of h, the goal is to identify the most probable sequence of future context elements $c_{n+1}, c_{n+2}, ..., c_{n+h}$. In the following, we describe the algorithmic approach to calculate these predictions based on the flow predictor.

Short-term prediction The starting point for short-term prediction is given by the current predictor state (h, a), from which the transition system is traversed. Algorithm 9 shows the steps involved in the calculation of the most probable next context.

For short-time prediction, we have to take into account that the same context c may be reached via different transitions from (h, a). Therefore, we have to sum the probabilities associated with each transition that is labelled with context c (line 2). Finally, the context returned as the prediction is the one with maximum probability, i.e., $c_{n+1} = \arg\max_{c \in C} P(X_{n+1} = c|(h, a))$ (line 4). The worst case time complexity is $O(|A| \cdot |C|)$ since the next context may potentially occur in each of the flow activities. However, we stress that, in practice, the search space is much more restricted by the flow structure. This structure only allows for a small subset of all activity-context combinations.

Algorithm 9 Short-Term Context Prediction

Require: current history state h
Require: current flow state $a = state(f)$
Ensure: $c_{n+1} = \arg\max_{c \in C} P(X_{n+1} = c|(h, a))$
1: **for all** $c \in C$ **do**
2: $Prob(c) \leftarrow \sum_{(h',a') \in \hat{S}} p((h, a), c, (h', a'))$
3: **end for**
4: **return** $\arg\max_{c \in C} Prob(c)$

Long-term prediction The calculation of most likely paths is known from Hidden Markov Models (HMMs) [62] where the Viterbi algorithm is used to discover paths of so-called hidden states for given observations. However, HMMs are based on predefined sets of states and constant transition probabilities. In our case, transition probabilities vary for each prediction horizon, and future paths depend on the initial state at the time of prediction and need to be explored. We address these issues in Algorithm 10.

The algorithm is based on an iterative approach that calculates the most likely path (sequence of context occurrences) of length h (prediction horizon). It starts with path length 1 and determines the most likely path of length i ($1 < i \leq h$) based on paths of length $i - 1$ (line 2-9). For this purpose, we compute $Prob_i(c)$, which denotes the probability of the most likely path of length i that ends in the occurrence of context element c. Initially, we set $Prob_0(c) = 1$ and, for all $i > 0$, $Prob_i(c)$ is calculated by adding the transition probabilities that reach c from paths of length $i - 1$ (line 4).

Based on $Prob_i(c)$, the sequence of context elements that define the most likely path is built incrementally. For this purpose, we associate with $path_i(c)$ the most likely sequence of context occurrences that ends in c for a path of length i. We then append to $path_i(c)$ the context that maximizes the path probability in each iteration (line 6), starting from a path length of $i = 1$.

For these calculations, we first define $reachable_k(h, a)$ as being the set of all predictor states reachable from state (h, a) through an arbitrary path of length k. That is, we initially have $reachable_0(h, a) = \{(h, a)\}$ and for all $k > 1$, $reachable_k(h, a)$ can be explored by following the outgoing transitions. Based on this, we define $R_k(c)$ as being the set of predictor states reachable through context c from any state in $reachable_{k-1}(h, a)$. More formally, we write

Algorithm 10 Long-Term Context Prediction

Require: current history state h
Require: current flow state $a = state(f)$
Require: prediction horizon h
Ensure: $c_{n+1}, c_{n+2}, ..., c_{n+h}$ most likely path
1: $i \leftarrow 1$
2: **while** $i <= h$ **do**
3: **for all** $c \in C$ **do**
4: $Prob_i(c) = max_{c' \in C}\{Prob_{i-1}(c') \cdot \sum_{(h,a) \in R_{i-1}(c'),(h',a') \in \hat{S}} p((h, a), c, (h', a')\}$
5: **if** $(i > 1)$ **then**
6: $path_i(c) \leftarrow$ append $\arg\max_{c' \in C}\{Prob_{i-1}(c') \cdot \sum_{(h,a) \in R_{i-1}(c'),(h',a') \in \hat{S}} p((h, a), c, (h', a')\}$
7: **end if**
8: **end for**
9: $i \leftarrow i + 1$
10: **end while**
11: $c^* = \arg\max_{c \in C} Prob_h(c)$
12: $path_h(c^*) \leftarrow$ append c^*
13: **return** $path_h(c^*)$

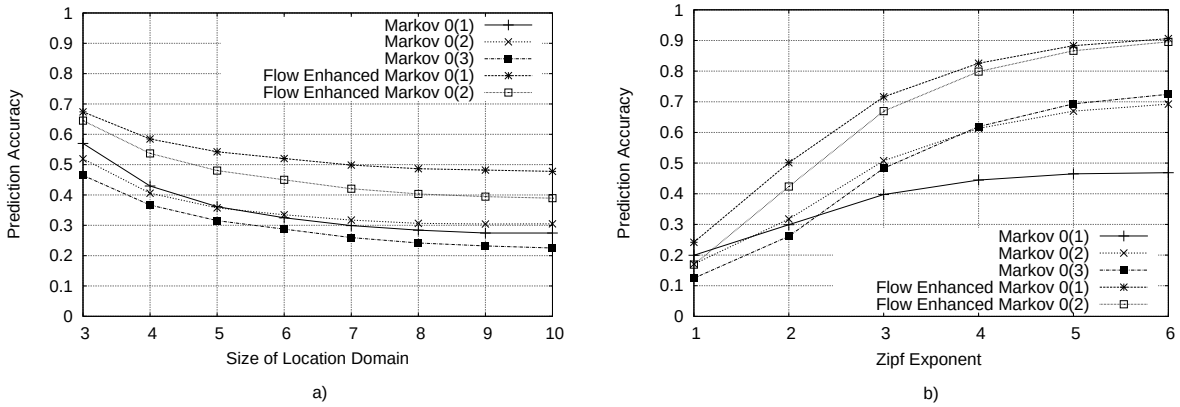


Figure 4.3: Short-Term Prediction Accuracies for Activity-Based Mobility Model with Parameter Settings a) Zipf exponent s=2 b) Location Domain $|L| = 7$

$$R_k(c) = \{(h', a') \in \hat{S} | \exists (h'', a'') \in reachable_{k-1}(h, a) : ((h'', a''), c, (h', a')) \in \tau\}$$

with (h, a) being the current state at which the prediction starts. $R_k(c)$ is used in the algorithm (line 4 and 6) to identify the context occurrences that lie on the most likely paths to c for each step of the iteration.

After the termination of the iteration, the most likely path for horizon h is the path that maximizes the path probability for a context $c^* \in C$. Consequently, we can return the stored path associated with the context $c^* = \max_{c \in C} Prob_h(c)$ (line 9). As the path ends with c^*, we have to append this context to the complete path up to horizon h. The iterative approach guarantees that the time complexity is bound by $O(h * (|A| \cdot |C|)^2)$ in the worst case. However, the search space will in reality be often restricted by the fact that not all context and activities are reachable from the current state.

4.2.4 Evaluation

We have implemented a simulation environment in order to evaluate the suitability of Adaptable Pervasive Flows for context prediction. We compare history-only predictors with their flow-enhanced counterparts based on synthetic context histories. This allows us to analyse the accuracies of the predictors for a spectrum of possible scenarios. We have implemented the *0(k)* family of Markov predictors as well as the flow enhanced-version of these. Although our approach is applicable to any form of discrete context, we study the accuracy of location prediction based on an activity-based mobility model that associates activities with locations as explained in the following.

First, we randomly generate flow models of different structure and size. Flow models are created from two different workflow patterns, i.e., sequences and branches, and form directed acyclic graphs. Second, we probabilistically associate each flow activity with locations from the domain $L = \{l_1, l_2, ..., l_n\}$. For each activity, we independently derive location visit probabilities based on a Zipf distribution, so that the probability to visit the i-th location during an activity is given by $P(X = i) = \frac{i^{-s}}{\sum_{n=1}^{|L|} n^{-s}}$. The exponent s allows us to vary the density of the visit probabilities. Based on the activity-based model of user mobility, we generate n flow-enhanced context histories $H_{f_1}, H_{f_2}, ..., H_{f_n}$ as sequential input for the predictors. We compare the different predictors based on an accuracy metrics, that is defined as the ratio of number of correct predictions to all predictions made. If a prediction is not possible due to the fact that the current context has not been learnt before, we count it as incorrect prediction. Predictions are determined simultaneously to the learning phase, i.e., after each predictor update we compute a prediction and validate it. A simulation run consists of a generated flow for which we create 100 context histories that describe possible executions of the flow. The results discussed in the following represent

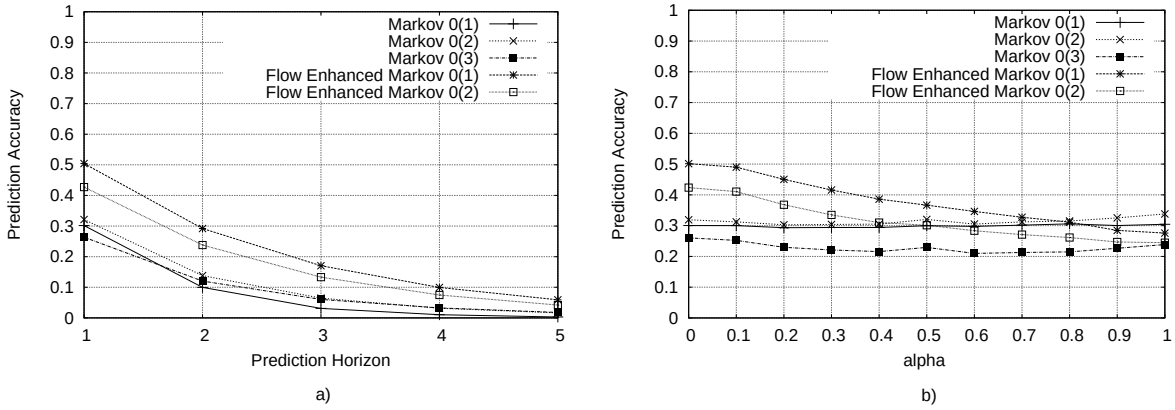


Figure 4.4: a) Long-Term Prediction Accuracies b) Prediction Accuracy for History-Generated Mobility Patterns

the average of 500 simulation runs for each measurement.

Figure 4.3 a) shows the short-term prediction accuracies for an increasing size of the location domain and Zipf exponent $s = 2$. Due to the higher uncertainty associated with a larger location domain, the prediction accuracy is negatively affected for all predictors. However, the flow-enhanced predictors outperform the history-based predictors for all of the evaluated sizes of L. Particularly, the relative improvement rises from 19% to 56% compared to the best history-based predictor for an increasing size of L. This illustrates the capability to resolve ambiguities from the history due to the available flow knowledge. Since we associate a single location visit with an activity in the simulation, higher-order Markov models do not improve the accuracy of the flow-enhanced predictors. As more states have to be learnt in this case, additional prediction misses are caused.

In Figure 4.3 b) we compare the predictors for a location domain of $|L| = 7$ and varying Zipf exponents. For increasing exponents the locations visit probabilities exhibit a highly skewed distribution, so that the predictors are able to deduce a higher fraction of correct predictions. However, the flow-enhanced predictors are able to capture patterns that remain hidden for the history-based predictors. The flow-enhanced predictors achieve a relative improvement of 25% in prediction accuracy compared to the best history-based predictor. Moreover, the results show that history-based Markov models can benefit from a larger memory for prediction if activities are more restricted to specific locations. In this case, longer sequences of past locations more accurately imply the next location. Nevertheless, still a substantial fraction of patterns can only be distinguished with flow knowledge.

Figure 4.4 a) depicts the results of long-term prediction for parameters of $|L| = 7$ and Zipf exponent 2. The absolute prediction accuracies for all predictors naturally decrease for higher prediction horizons. Particularly, for horizon 5 the absolute accuracy has reached a degree, where no sensible predictions can be made any more. However, the relative improvement in accuracy of flow-based prediction compared to the best history predictor monotonically increases from 57 % for horizon 1 to considerable 331 % for horizon 5. Consequently, especially long-term predictions can benefit from our enhanced context prediction. In the next step, we extend the simulation model with the possibility to include history-generated patterns in the context histories. The history patterns are generated based on a $0(2)$ Markov source that is trained from the location traces of the activity-based model. This model naturally favours history-based predictors due to the underlying Markov assumption. We introduce the parameter α that allows to vary between both models, i.e., α indicates the portion of the context history which is generated by the Markov source and the portion $(1 - \alpha)$ which adheres to the activity-based model. Figure 4.4 b) shows a monotonic decrease of the prediction accuracies of the flow-based predictors for increasing values for α, while the history-based predictors remain constant. For $\alpha \geq 0.8$ the best flow-based predictor even performs worse than the *0(1)* Markov predictor. The reason is that, due to the correlation with flow activ-

ities, history-based patterns are scattered over many states of the flow predictor. The consequence is that the patterns cannot be learnt as fast as in the case of a classical history predictor, and more training data is necessary to achieve the same accuracy. We will address this issue in future work by the design of a hybrid prediction scheme, that involves components of both predictors and only utilizes flow knowledge for patterns that cannot be discovered by history-based predictors.

4.3 PreCon – Expressive Prediction using Stochastic Model Checking

Figure 4.5 gives a high-level overview of the PreCon approach. We assume that a context recognition system monitors the context of the user and records *context traces* (time-stamped series of consecutive context changes) in user histories. For instance, a context trace may contain information about which activities have been executed at what time and location. The context traces are given as input to our *learning algorithm*, which processes them to obtain an SUM. Our framework allows context traces to be processed either in a *batch-like* fashion or in *real-time*. For the batch-like approach, the learning algorithm processes one or more context trace and creates a new SUM, while for the real-time approach, the algorithm updates the existing SUM each time a relevant context change has been observed. In both cases, the information about sequential changes in user context is used to build a *Semi-Markov Chain* (SMC) – a well-known stochastic process model that we use to represent SUMs. A SMC is a probabilistic state transition system that maintains the discrete states of the user behaviour and the associated state transition probabilities. Furthermore, the temporal characteristics of context changes (the so-called *dwell times*) are modelled by the SMC. This is the key to PreCon's concept of time-dependent queries.

Applications can specify *context prediction queries* using different temporal operators that are part of a temporal stochastic logic. This query language provides well-defined semantics to express *reachability properties* (e.g. will the user arrive at a certain location) and *invariant properties* (e.g. will the user stay at a certain location). A context prediction query is evaluated on an SMC to calculate the probability with which the specified properties hold. A querying application can specify a probability threshold with which the resulting probability is compared, and a *true* or *false* is returned depending on the outcome. Finally, the querying application may use this result to take proactive decisions in terms of, e.g. user interaction and context-aware services, to enhance the user's experience. In the following sections, we will investigate each element of PreCon in turn.

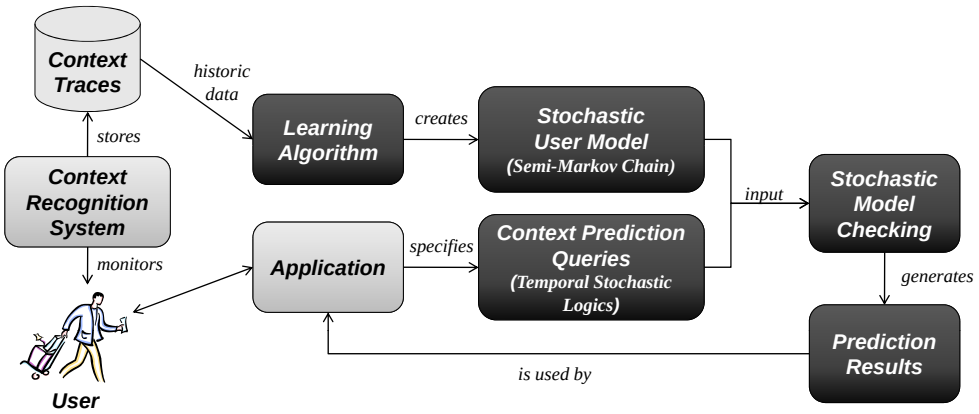


Figure 4.5: Overview of the PreCon approach – Concepts specified by PerCon are shown in dark boxes

4.3.1 Stochastic User Model

People follow varying behavioural patterns in their daily lives, such that real world user behaviour can not be described in a deterministic way. Consequently, we require a user model that is able to deal with this probabilistic nature of human behaviour. In the following, we give a precise formal definition of the SUM. Subsequently, we present our approach for learning such a SUM from observations recorded in real-world context traces.

4.3.1.1 Semi-Markov Chain

We represent an SUM as a Semi-Markov Chain (SMC) [39]. In general, Markov Chains are a popular means for describing stochastic processes with discrete state spaces. In addition to that, SMCs specify a so-called *state dwell time* – an arbitrary probability distribution that is associated with every state transition specifying the amount of time spent in a given state. Formally, a SMC M is a 3-tuple defined as:

$$M = (S, p, q)$$

where S is the state space, $p : S \times S \rightarrow [0, 1]$ with $\forall s \in S : \sum_{s' \in S} p(s, s') = 1$ is the transition probability function, and $h : (s, s', t) \mapsto [0, 1]$ with $t \in \mathbb{R}^+$ represents the distribution of dwell times associated with a state transition $(s, s') \in S \times S$. For $h : (s, s', t)$, we will also write $h_{s,s'}(t)$ for brevity reasons. The SMC allows us to describe a user's behaviour in the following manner: At each point in time, a user is in a state $s \in S$ that is identified by his current context (cf. Section 4.3.1.2). While the user acts in the real world, his context changes and his SMC moves to a new state $s' \in S$ representing the new context. s' is called the *successor state* of s, and s' is visited with a certain probability $p(s, s')$. Before leaving the current state s, s is active for a limited amount of time (the dwell time represented by $h_{s,s'}(t)$). During this time period the user's context does not change.

4.3.1.2 Learning a SMC

In contrast to classical model checking, we do not expect a designer of the system to define the SMC underlying the real world behaviour. Instead, we apply a *learning approach* and derive the SMC from the observations of a context recognition system. In the following, we describe the basic elements of an SMC as well as the procedure of how to process context observations in order to learn an SMC.

User States A user state $s = (c_1, ..., c_n)$ is an n-dimensional vector of context information. Each component c_i of s is of a specific *context type* C_i, and $C_1, ..., C_n$ are the context types known to the system with domains $Dom(C_1), ..., Dom(C_n)$. E.g. c_i may be an integer value from $Dom(C_i) = \mathbb{N}^+$, and C_i may be the *ambient temperature* type. Other types could be *location* and *activity*, and the corresponding domains could be enumerations of possible activities and symbolic location identifiers respectively. For example, the state $s = (\text{"}meeting\ room\text{"}, \text{"}give\ presentation\text{"}) \in Dom(Location) \times Dom(Activity)$ describes the fact that the user executes the activity *give presentation* in a location referred to as *meeting room*. Whenever a combination of context information $(c_1, ..., c_n)$ is detected that has not already been encountered for the specific user, a new user state $s = (c_1, ..., c_n)$ is added to the SMC.

Transition Probabilities A concrete series of consecutive user states is represented as a stochastic process of random variables $X_1, X_2, X_3, ...$, where X_i refers to the state occupied after the i-th state transition. In order to learn the state transition probabilities, we assume the Markov property: The probability $p(s, s')$ for the state s' to be visited next only depends on the current state s, and is independent of all previous state changes. This assumption can be extended such that $p(s, s')$ depends on the k last visited states (k-order Markov models [70]) if needed, and PreCon operates on these more general k-order models. However, for simplicity, we assume $k = 1$ here. The math is essentially the same.

Assuming a stationary probability distribution, the probabilities $p(s, s')$ can be estimated from the history of past state transitions: Let $w_{s,s'}$ be the *transition weight*, which denotes the number of transitions from s to s' as observed in the history. The transition probability $p(s, s')$ is defined as $p(s, s') = P(X_{n+1} = s' | X_n = s) = \frac{w_{s,s'}}{\sum_{s'' \in S} w_{s,s''}}$. Thus, the probability is the ratio of the number of observed state transitions from s to s' to the number of all observed transitions from s.

Dwell Time Distribution The dwell time in state s is modelled as a random variable D_s. We learn the probability distribution of D_s conditioned on each transition such that $h_{s,s'}(t) = P(D_s = t | X_{n+1} = s', X_n = s)$. For this purpose, we observe the time periods that pass between consecutive changes in user state. In order to limit the storage and computation overhead, we apply a discretization and divide time into intervals of equal size Δt, such that the i-th time interval is defined as $I_i = [i \cdot \Delta t, (i + 1) \cdot \Delta t)$. The distribution $h_{s,s'}$ can then be derived as follows: Let $w^i_{s,s'}$ be the number of transitions (s, s') that occurred in the interval I_i such that $w_{s,s'} = \sum_i w^i_{s,s'}$ is the total number of observed transitions (s, s'). Then the probability for spending exactly time t in state s before leaving to successor state s' is calculated as

$$h_{s,s'} : t \mapsto \frac{w^{\lfloor \frac{t}{\Delta t} \rfloor}_{s,s'}}{\Delta t \cdot w_{s,s'}}. \tag{4.1}$$

In equation 4.1, we use Δt as a normalization factor to ensure that $\int_0^\infty h_{s,s'}(t)\, dt = 1$ for the cumulative distribution. As we deal with a discrete representation of the dwell time distribution, the cumulative distribution function $\int_a^b h_{s,s'}(t)\, dt$ is computed as a sum of intervals over the probability mass function. More precisely, the cumulative probability is determined by the sum over the intervals which are enclosed by the integral ranges between a and b. As a and b may fall into discretization intervals, we interpolate the probability associated with the fraction of the corresponding intervals based on a linear function. The cumulative distribution is later used in Section 4.3.3 to determine the probability resulting from time-bounded queries.

4.3.2 Prediction Query Language

PreCon's prediction query language is based on *Continuous Stochastic Logic* (CSL) [8], a probabilistic derivative of branching-time temporal logics (applied in classical model checking). CSL provides operators for verifying *temporal properties* of probabilistic state transition systems. Applications construct queries from these *temporal operators* and submit them to PreCon. The operators are then evaluated on the learnt SMC to verify the specified properties.

The state space S can be traversed by going from one state to the next as the transitions among the states permit. The resulting series of visited states (called a *path*) models one possible temporal behaviour of the user. For a context prediction, PreCom starts at the state $s \in S$ the user currently occupies in the real world and evaluates the given query from there, possibly considering all possible paths starting at s (depending on the temporal operators in the query). The query language is defined as follows:

Let $p \in [0, 1]$ be a probability threshold, let $\triangleleft \in \{\leq, \geq\}$ be a comparison operator, let $t \in \mathbb{R}^+$ be a time bound, and let $(C_i, c \in Dom(C_i))$ be a contextual value c of type C_i. Queries can be composed from CSL using the following grammar:

A query is a temporal-logic formula Φ with

$$\Phi = true \mid (C_i, c) \mid \Phi \wedge \Phi \mid \neg\Phi \mid \mathbb{P}_{\triangleleft p}(\varphi),$$

where φ is a *path formula* defined as

$$\varphi = X^{\leq t}\Phi \mid F^{\leq t}\Phi \mid G^{\leq t}\Phi \mid \Phi_1 U^{\leq t} \Phi_2$$

Using CSL, we can investigate *reachability properties* (using operators X and F) and *invariant properties* (using operators G and U) of future user behaviour. X is the *Next* operator. It evaluates a condition Φ on all immediate successor states of the current user state s. Φ is expressed as a name-value pair (C_i, c) consisting of the name of a context type C_i (e.g. *location*) and a specific context value c (e.g. *office*). The query "Will the next location be the office?" can be expressed by applying the *Next* operator to $\Phi = (location, office)$, resulting in $X(location, office)$. F is the *Eventually* operator and can be used to verify if a condition Φ holds in any state reachable from s through paths in the SMC. G is the *Globally* operator and can be used to check if the condition Φ holds in every state on all paths starting in s. U is the *Until* operator and expresses that eventually Φ_2 must hold and Φ_1 must hold on all paths starting at the current state until Φ_2 holds.

Time is a first order construct of the prediction query language. All operators are associated with a time constraint t, defining an upper bound on the time, which may pass until the desired property holds. Having such a time bound and using the dwell time distributions to evaluate time-bounded queries enables us to formulate time-dependent queries.

The *raw* predictions are always probabilistic in nature when a query is evaluated. So the answer of the model checking algorithm is of the form "The user enters his office within the next 10 minutes with probability 0.74". A querying application, however, usually expects a *true* or *false* as an answer. Therefore, the calculated probabilities are compared to a probability threshold p, which is expressed in the subscript of a query formula ($\mathbb{P}_{\triangleleft p}(\varphi)$). The querying application specifies this probability and gets a boolean result depending on whether the outcome of the query evaluation exceeds the threshold or not[1].

In Table 4.1, we give some examples for behavioural properties which can be expressed as CSL formulas. The examples demonstrate the range of different use cases for context predictions, including queries with different semantics and context types.

Query	**Explanation**
$\mathbb{P}_{\geq 0.8}(X^{\leq 10min}(location, office))$	Will the office be the uer's next location within the next 10 minutes with a probability of ≥ 0.8?
$\mathbb{P}_{\geq 1.0}$ $(location, home)$ $\wedge$ $\mathbb{P}_{\geq 0.8} F^{\leq 30min}(\neg(location, home))$	Is the user currently at *home* and will he *eventually* leave with a probability ≥ 0.8 within the next 30 minutes ?
$\mathbb{P}_{\geq 0.6}$ $G^{\leq 30min}$ $(activity, walking)$	Will the user be *walking* within the next 30 minutes with a probability ≥ 0.6?
$\mathbb{P}_{\geq 0.2}$ $((location, stuttgart)$ $U^{\leq 60min}$ $(location, home))$	Will the user be in Stuttgart with a probability ≥ 0.2 until he eventually reaches his *home* within the next hour ?
$\mathbb{P}_{\geq 1.0}(activity, biking)$ $\wedge$ $\mathbb{P}_{\geq 0.8}(F^{\leq 60min}$ $(location, home)$ $\wedge$ $(activity, sitting))$	Is the user currently biking (anywhere) and will *eventually* relax ($activity$ = *sitting*) at his home with a probability ≥ 0.2 within the next hour?

Table 4.1: Examples of context prediction queries

The probability threshold p is an application-dependent value to influence the trade-off between *false positives* (queries that evaluate to *true* but prove to be false later on) and *false negatives* (queries that evaluate to false but actually become true in reality): A higher threshold reduces the number of false positives and increases the number of false negatives. A lower threshold has the opposite effect. Consequently, the concrete threshold defines the ratio of false negative and false positives that the application is willing to accept. The choice for the threshold is dependent on the application semantics. For example, in security critical applications it is usually beneficial to prepare for exceptional cases even if they might not occur. Hence, such applications may tolerate a higher number of false positives rather than false negatives. On the contrary, a large number of false positives may negatively impact the satisfaction of a user in an application that delivers advertisements based on his predicted future location. In this case

[1] Applications can also access the raw prediction result in case they require more complex threshold comparisons.

a higher probability threshold is beneficial to prevent the user from being overwhelmed by irrelevant advertisements.

4.3.3 Query Processing

Classical model checking algorithms assume static state transition systems, where the system is analysed at design-time and behavioural properties are only studied at state entry times. In our case, the system that is subject to the verification is dynamic. In particular, the probability resulting from the evaluation of a query is depending on the time Δd, that has passed since the current state s was entered. Therefore, we have to extend the standard model checking approach to account for Δd (referred to as the *running dwell time* in the following) by devising new ways of evaluating the temporal operators X and U. This is sufficient since it can be shown that, F and G can be expressed using the X and U operators [9]. For example, the reachability property $F^{\leq t}\Phi$ can be transformed to the equivalent expression ($trueU^{\leq t}\Phi$). We refer to X and U as the *basic operators* in the following. Arbitrarily complex temporal-logic formula can be evaluated in a bottom-up manner based on a tree representation [9] using only the basic operators. Thus, evaluating a query requires two things:

1. We need to be able to determine whether a given state s satisfies a basic context constraint $\Phi = (C_j, c)$. The basic satisfaction relation is defined as $(s = (c_1, ...c_j, ..., c_n) \models \Phi) \Leftrightarrow c_j = c$.

2. We need the ability to calculate the probability of $X^{\leq t}\Phi_1$ and $\Phi_2 U^{\leq t}\Phi_1$ for some basic context constraints Φ_1, Φ_2. Intuitively speaking, this involves calculating the probabilities of reaching a state s with $s \models \Phi_1$ and of traveling a path where $s_i \models \Phi_2$ holds for every state s_i.

Our model checking problem can be solved by evaluating a satisfaction relation $\models$ for the path formula φ enclosed by the probabilistic operator $\mathbb{P}_{\triangleleft p}(\varphi)$ as follows:

$$(s, \Delta d) \models \mathbb{P}_{\triangleleft p}(\varphi) \Leftrightarrow P(s, \Delta d \models \varphi) \triangleleft p$$

In other words, the path formula φ is satisfied after Δd time units have passed in state s iff the probability $P(s, \Delta d \models \varphi)$ for the occurrence of φ satisfies the threshold condition $\triangleleft p$.

In the following, we will present the evaluation approach for the two basic operators in detail. Let i be the index of the last state transition that was observed, such that $X_i = s$ denotes the current state and $D_s = \Delta d$ is the current dwell time that has passed in this state. As a common basis for the computations, we define the probability for moving from the state s to a successor state s' within time t using the information present in the SMC as follows:

$$P\left(X_{i+1} = s', D_s \leq \Delta d + t | X_i = s, D_s > \Delta d\right) \tag{4.2}$$

$$= \frac{P\left(X_{i+1} = s', \Delta d < D_s \leq \Delta d + t | X_i = s\right)}{\sum_{s' \in S} P\left(X_{i+1} = s', D_s > \Delta d | X_i = s\right)} \tag{4.3}$$

$$= \frac{p(s, s') \cdot \int_{\Delta d}^{\Delta d + t} h_{s,s'}(x)\, dx}{\sum_{s' \in S} p(s, s') \cdot \int_{\Delta d}^{\infty} h_{s,s'}(x)\, dx} \tag{4.4}$$

We use Baye's rule to transform formula (4.2) into (4.3), which is free of the dwell time distribution in the conditional probability. Thus it can be computed using the state transition probabilities and the dwell time distribution present in the SMC (Equation 4.4).

4.3.3.1 Next Operator *X*

The next operator limits the search space for the satisfaction of property φ to the immediate successor states of the current state s. Due to the running dwell time, we have to consider the dwell time distribution only from the time Δd onwards and express this using a subscript in $X^{\leq t}_{\Delta d}$. We extend the model checking approach given by Lopez et al. [51] accordingly, as follows:

$$P(X^{\leq t}_{\Delta d}(\varphi)) \tag{4.5}$$

$$= \sum_{s' \in S \wedge s' \models \varphi} P\left(X_{i+1} = s', D_s \leq \Delta d + t | X_i = s, D_s > \Delta d\right) \tag{4.6}$$

$$= \frac{\sum_{s' \in S \wedge s' \models \varphi} p(s,s') \cdot \int_{\Delta d}^{\Delta d + t} h_{s,s'}(x)\, dx}{\sum_{s' \in S} p(s,s') \cdot \int_{\Delta d}^{\infty} h_{s,s'}(x)\, dx} \tag{4.7}$$

We can calculate $P(X^{\leq t}_{\Delta d}(\varphi))$ directly, using Equation 4.4. The denominator is the probability of reaching arbitrary next states in greater than Δd time units, whereas the nominator reduces this probability to states where the φ holds. This ratio represents the desired probability considering the running dwell time Δd.

4.3.3.2 Until Operator *U*

For the until operator, the satisfaction of the property φ must be evaluated along all paths which can be reached from the current state within the given time bound. The exploration of the state space is therefore not necessarily bound by the immediate successor states. Again, we extend the approach of Lopez et al. [51] by additionally considering the running dwell time Δd. This is expressed using a subscript in $U^{\leq t}_{\Delta d}$. The probability for the satisfaction of $U^{\leq t}_{\Delta d}$ can then be calculated as follows:

$$P(\Phi_1 U^{\leq t}_{\Delta d} \Phi_2) = F_a(s, s', t, \Delta d) \tag{4.8}$$

$$F_a(s,s',t,\Delta d) = \begin{cases} 1, \text{ if } s \models \Phi_2 \\ \frac{1}{\sum_{s' \in S} p(s,s') \cdot \int_{\Delta d}^{\infty} h_{s,s'}(t) dt} \cdot \sum_{s' \in S} \int_{\Delta d}^{\Delta d + t} p(s,s') \\ \qquad \cdot h_{s,s'}(x) \cdot F_b(s,s',t-x) dx, \text{ if } s \models \Phi_1 \wedge \neg \Phi_2 \\ 0, \text{ otherwise} \end{cases} \tag{4.9}$$

$$F_b(s,s',t) = \begin{cases} 1, \text{ if } s \models \Phi_2 \\ \sum_{s' \in S} \int_0^t p(s,s') \cdot h_{s,s'}(x) \cdot F_b(s,s',t-x) dx \\ \qquad \text{, if } s \models \Phi_1 \wedge \neg \Phi_2 \\ 0, \text{ otherwise} \end{cases} \tag{4.10}$$

The extension of the standard algorithm results in two functions F_a (4.9) and F_b (4.10). Function F_a is used in the first step of the verification, starting from the current user state s taking the running dwell time in s into account. Function F_a uses F_b for calculating the probability over all the possible paths starting at s. F_b calculates the probabilities recursively using convolution of dwell time distributions, taking into account that a state may be left at each point in time within the remaining time horizon.

4.3.4 Evaluation

We have evaluated PreCon using real-world context traces from a case study in a German geriatric nursing home. The nursing home is an intensive care station for elderly people suffering from dementia

and other old-age diseases. The patients are accommodated in rooms on a nursing ward, where they receive care from nurses throughout the day. Each nurse visits patients in different rooms and performs treatment activities (e.g., the patient morning hygiene). PreCon predicts the future context of the nurses, in order to optimize the tasks scheduled by an intelligent workflow management system. The integration of context prediction into the scheduling decisions is part of the European research project ALLOW [37]. In order to obtain context traces, the nurses were accompanied over the course of 25 days during 3-5 hours in the morning shift. The traces consist of time-stamped entries of (1) the activities performed by nurses (2) the locations of their visits and (3) the ids of the patients they took care of. Thus, the records define a time series of multi-dimensional context, where each entry denotes a discrete change of context associated with a nurse. Given the context traces, PreCon learns a SMC to represent the behaviour of each nurse. In order to evaluate the impact of different context types on the prediction outcome, we varied the types of context used to learn the SMCs. We investigated three different state spaces, i.e., $s \in Dom(Location)$, $s \in Dom(Location) \times Dom(Activity)$ and $s \in Dom(Location) \times Dom(Activity) \times Dom(Patient)$.

It is important to note that comparisons with existing context prediction systems are not possible at this time as they cannot produce the type of temporal predictions generated by PreCon. Since PreCon is the first system to venture into this area, we use metrics from the area of information retrieval (as explained in the following) to asses the general performance of PreCon.

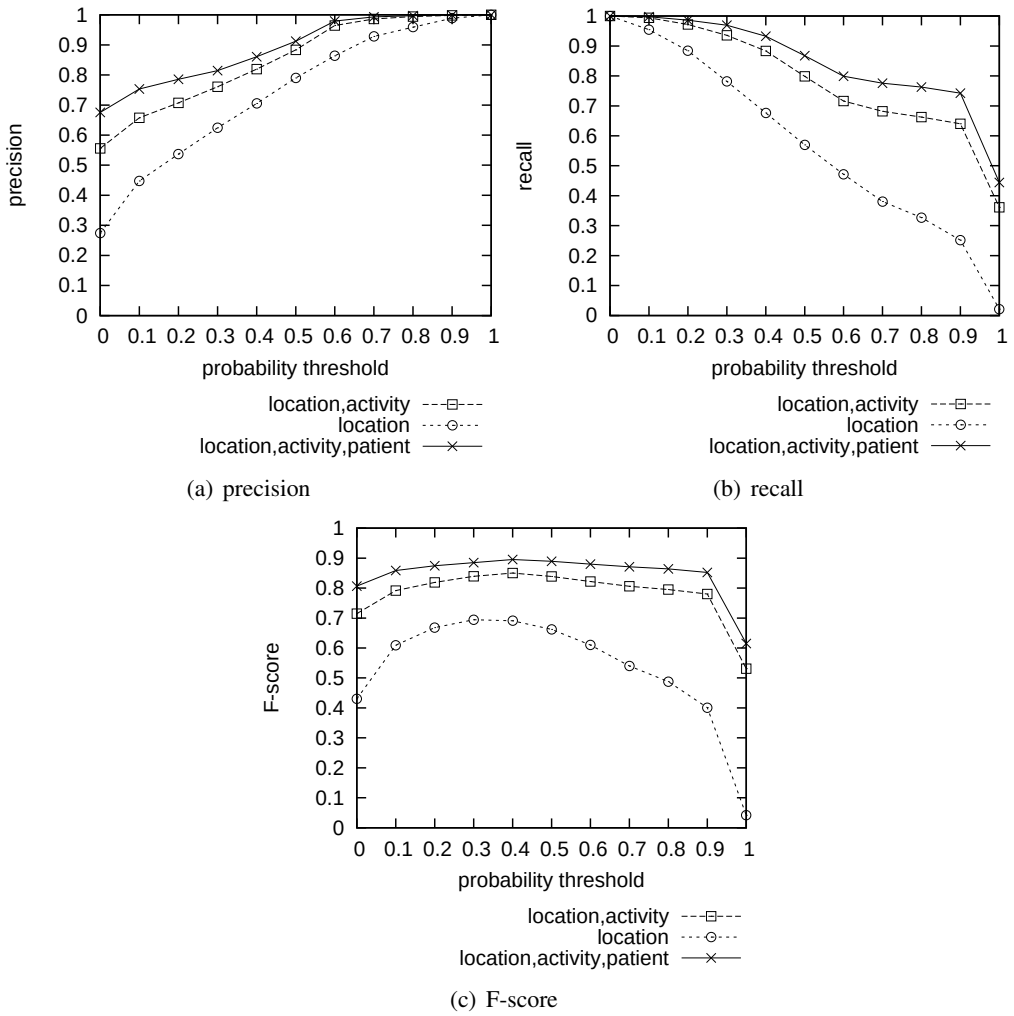
4.3.4.1 Metrics

We performed the evaluations using the metrics *precision*, *recall* and *F-score* known from the area of information retrieval. If a query result exceeds the probability threshold, we count it as either *true positive* (TP) or *false positive* (FP), depending on whether the prediction matches the real-world context. Otherwise, in case the prediction remains below the probability threshold, we distinguish between *true negatives* (TN) or *false negatives* (FN). We count the occurrences of (TP), (FP), (TN), and (FN) in order to calculate the metrics. This way, we can evaluate the influence of a varying probability threshold p on the *precision* defined as $\frac{TPs}{TPs+FPs}$ as well as on *recall* defined as $\frac{TPs}{TPs+FNs}$. While precision is a measure of the exactness of the predictions, recall is used to quantify the completeness of the predictions. Additionally, we evaluate the *F-score* which gives a combined measure of both and is defined as $2 \cdot \frac{\text{precision} \cdot \text{recall}}{\text{recision+recall}}$. The performance of these metrics gives important insight in how proactive applications are affected by a choice of the probability threshold as will be discussed in the next subsection for different queries.

4.3.4.2 Basic Queries

We evaluated our approach for two exemplary queries $\mathbb{P}_{\triangleleft p}(X_f^{\leq t}(\varphi))$ and $\mathbb{P}_{\triangleleft p}(\ F^{\leq t}\Phi_2) = \mathbb{P}_{\triangleleft p}(\ trueU^{\leq t}\Phi_2)$. We generated queries for the future location of a user. The time constraint associated with these queries is set to $t = 10$ minutes. We evaluated the queries repeatedly, i.e, predictions were computed upon a state change and periodically after $\Delta E = 10$ seconds have passed in a state. The results discussed in the following show the average of 2000 query evaluations.

Figures 4.6(a)-(c) illustrate the results for the *Next* operator $X_f^{\leq t}(\varphi)$. Figure 4.6(a) shows the impact of a varying probability threshold on the precision metrics. As expected, the precision gains from an increase of the probability threshold. The reason is that the number of FP decreases because an increasing portion of the predictions with a low probability is discarded. At the same time, we can observe the highest precision if states are composed of the multi-dimensional context "concrete activities", "location" and "patient". In contrast, the precision remains lowest when states only contain location information (single-dimensional). Hence, the evaluation results show that additional context is relevant to discriminate user states such that more accurate predictions can be expected. Figure 4.6(b) shows the recall as a function of the probability threshold. The results are reciprocal to the precision results: For an increasing probability threshold the number of FN increases, as more and more predictions are discarded that



(a) precision

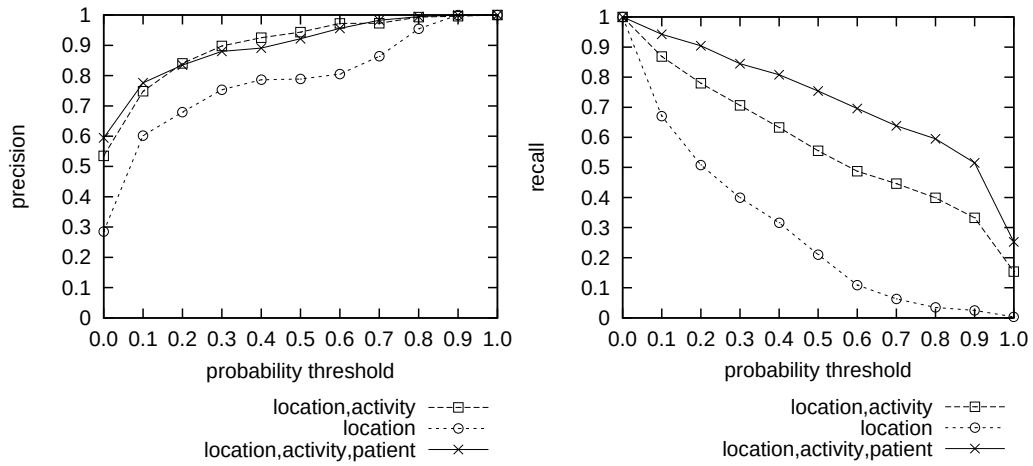
(b) recall

(c) F-score

Figure 4.6: Evaluation results for the Next operator

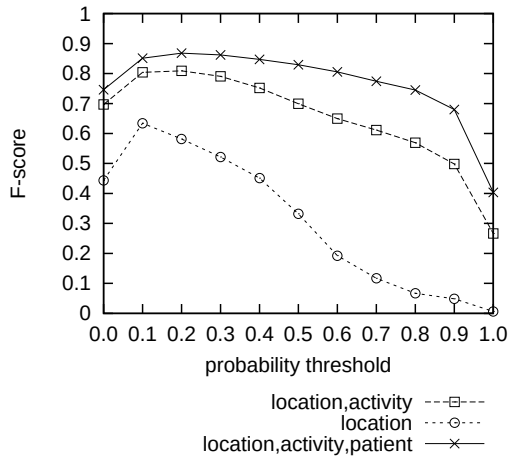
actually match the future behaviour. This result illustrates the trade-off between precision and recall for different values of the probability threshold: If the threshold is increased to guarantee more reliable predictions, the risk of discarding correct predictions with a low probability rises. Figure 4.6(c) shows the F-score and reveals the characteristics of this trade-off: The F-score rises as the threshold increases from 0 to 0.4. Up to this threshold, the gain from a higher precision outweighs the loss in recall. However, for a threshold higher than 0.4, the loss due to the loss in recall becomes more significant, so that the score is negatively affected. For applications that are interested in a good trade-off, we therefore recommend $p = 0.4$ as a probability threshold. In this case, we achieve a F-score of 0.86, which indicates a very good performance.

Figures 4.7(a)-(c) depict the evaluation results for the until operator. Figure 4.7(a) shows that the precision significantly gains already for lower thresholds $p > 0$. The reason is that Φ_2 can be fulfilled in all states reachable within the time constraint. Thus, the chance to encounter the expected context is increased. In contrast, the same chance is limited to the state successors in case of the next operator. At the same time, the recall is highly reduced by an increasing threshold as shown in Figure 4.7(b). Since the evaluated queries also address future context, which appears in states reachable over multiple transitions that are uncertain to occur, a significant amount of predictions has only a minor probability. This large number causes a lot of FNs in total, so that a significant amount of correct predictions is discarded. This observation is also reflected in Figure 4.7(c), which shows the F-score. For a threshold $p > 0.2$ the high loss in recall dominates the gain in precision. Compared to the next operator, the until operator is more



(a) precision

(b) recall



(c) F-score

Figure 4.7: Evaluation results for the Until operator

sensitive to the choice of the probability threshold. We therefore recommend $p = 0.2$ as a threshold for applications to deal with the inherent trade-off for the until operator. In this case, a good result with an F-score of 0.87 can be achieved.

4.4 Summary and Conclusions

In Section 4.2, we have presented a new context prediction scheme that is able to provide history predictors with domain-specific knowledge inherent to flow. Our context predictor learns the relationship of flow activities with context changes observed in the real world. We represent this relationship as a probabilistic state transition system which is incrementally refined from the execution of flows at runtime. For context prediction, we traverse the state space of possible context changes to determine the most likely paths of future context occurrences.

In our evaluation, we have shown that the inclusion of knowledge about user activities in the prediction model significantly improves the prediction accuracy, as classical predictors are limited by their agnostic view on the application domain.

In Section 4.3, we presented PreCon, a novel approach to rendering flow-based context prediction more expressive than existing systems. In PreCon, user behaviour is represented by Semi-Markov Chains (SMC), and temporal-logics is used as a query language. We extended well-known model-checking

techniques to deal with the online character of context predictions and to allow for continuous learning of SMCs. PreCon's query language provides a powerful means for applications to pose temporal queries for reachability and invariant properties of future context. Thus, PreCon goes far beyond existing approaches and represents a new class of context prediction systems that enable intelligent ubiquitous applications to take much more educated decisions.

We evaluated PreCon based on a real-world case study in the area of healthcare using metrics from information retrieval and showed that it exhibits a good performance. Moreover, our evaluations yielded indications for choosing sensible parameters for different classes of applications.

Overall, this represents a notable step forward from existing context prediction systems. We have shows that flow knowledge can be used to improve prediction capabilities. This means that flow-based applications have a great potential to be proactive and run in the background, unnoticed by the user. The feature of time-based queries provides applications and the flow system itself with a powerful means to pose meaningful queries.

Chapter 5

Conclusions

In this document, we have discussed three important elements of flow control in pervasive adaptable flows: (1) flow distribution, (2) robust flow navigation and (3) context prediction. We have developed these concepts over the last 18 months of the project.

The area of flow distribution is specifically tailored to deal with changing performance and quality of service in flow-based applications. These are new issues that arise from the fact that we consider flows in dynamic mobile environments, and the concepts we developed represent an important step towards a cost-efficient and seamless integration of business processes and pervasive computing, an area that has been gaining momentum as the project was running.

The area of robust flow navigation cannot merely be seen as a tool for enabling flows as we envisioned them in the project. This research goes beyond the original goals of the project by demonstrating how context-aware systems as such can benefit from the concept of flow-based applications. In order to achieve robustness in context-aware systems, application knowledge is required to deal with the uncertainties involved with real-world context data and the means for measuring this data. Flows have turned out to readily provide this knowledge. They represent a temporal model of activities and the conditions under which transitions occur between them.

In the area of context prediction, we have been able to go way beyond the current state-of-the-art, basically by exploiting the same type of flow knowledge. Before the work described in this document, context prediction systems where very simplistic, allowing only for discrete projections of the next context state. Our work was the first to introduce application knowledge in order to improve the accuracy of predictions. Moreover, we were the first to apply model checking mechanisms in order to allow for much more powerful prediction queries. This new type of queries is directly usable and meaningful for applications and the end user since it incorporates real-time semantics.

We conclude that overall, the achieved robustness and proactivity represents a major step beyond the current state of the art in pervasive applications.

Bibliography

[1] W. M. Aalst, M. Adams, A. H. Hofstede, M. Pesic, and H. Schonenberg. *Flexibility as a Service*, pages 319–333. Springer-Verlag, Berlin, Heidelberg, 2009.

[2] O. Adam, O. Thomas, and G. Martin. Fuzzy WorkflowsEnhancing Workflow Management with Vagueness. In *EURO/INFORMS Istanbul 2003 Joint International Meeting*, pages 6–10, 2003.

[3] Otmar Adam and Oliver Thomas. A fuzzy based approach to the improvement of business processes. In *First International Workshop on Business Process Intelligence (BPI05)*, pages 25–35, September 2005.

[4] Otmar Adam, Oliver Thomas, and Dominik Vanderhaeghen. Fuzzy-set-based modeling of business process cases. In *ICCBR Workshops*, pages 251–260, 2005.

[5] B. Altakouri, G. Kortuem, A. Grunerbl, K. Kunze, and P. Lukowicz. The benefit of activity recognition for mobile phone based nursing documentation: A wizard-of-oz study. In *Wearable Computers (ISWC), 2010 International Symposium on*, 2010.

[6] Theodoros Anagnostopoulos, Christos Anagnostopoulos, Stathes Hadjiefthymiades, Miltos Kyriakakos, and Alexandros Kalousis. Predicting the location of mobile users: a machine learning approach. In *Proc. of the 6th Intl. Conf. on Pervasive Services*, 2009.

[7] Michael Armbrust, Armando Fox, Rean Griffith, Anthony D. Joseph, Randy Katz, Andy Konwinski, Gunho Lee, David Patterson, Ariel Rabkin, Ion Stoica, and Matei Zaharia. Above the clouds: A berkeley view of cloud computing. Technical report, University of California at Berkeley, February 2009.

[8] Christel Baier, Boudewijn Haverkort, Holger Hermanns, and Joost-Pieter Katoen. Model-checking algorithms for continuous-time markov chains. *IEEE Transactions on Software Engineering*, 29:524–541, 2003.

[9] Christel Baier and Joost-Pieter Katoen. *Principles of Model Checking*. MIT Press, 2008.

[10] Niranjan Balasubramanian, Aruna Balasubramanian, and Arun Venkataramani. Energy consumption in mobile phones: a measurement study and implications for network applications. In *Proc. of the 9th ACM SIGCOMM conference on Internet measurement conference (IMC)*, 2009.

[11] Luciano Baresi, Andrea Maurino, and Stefano Modafferi. Workflow partitioning in mobile information systems. In *Proc. of the IFIP TC8 working conference on Mobile Information Systems (MOBIS)*, 2004.

[12] T.S. Barger, D.E. Brown, and M. Alwan. Health-status monitoring through analysis of behavioral patterns. *Systems, Man and Cybernetics, Part A: Systems and Humans, IEEE Transactions on*, 35(1):22 – 27, 2005.

[13] Thomas Bauer and Peter Dadam. Efficient distributed workflow management based on variable server assignments. *Lecture Notes in Computer Science*, 1789/2000:94–109, 2000.

[14] Amiya Bhattacharya and Sajal K. Das. Lezi-update: An information-theoretic approach to track mobile users in pcs networks. In *Proc. of the 5th Annual ACM/IEEE Intl. Conf. on Mobile Computing and Networking*, 1999.

[15] A. Billionnet, M. C. Costa, and A. Sutter. An efficient algorithm for a task allocation problem. *J. ACM*, 39(3):502–518, 1992.

[16] Jit Biswas, Andrei Tolstikov, Maniyeri Jayachandran, Victor Foo Siang Fook, Aung Aung Phyo Wai, Clifton Phua, Weimin Huang, Louis Shue, Kavitha Gopalakrishnan, and Jer-En Lee. Health and wellness monitoring through wearable and ambient sensors: exemplars from home-based care of elderly with mild dementia. *Annales des Télécommunications*, 65(9-10):505–521, 2010.

[17] A. F. Bobick and Y. A. Ivanov. Action recognition using probabilistic parsing. In *Proc. of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 1998.

[18] Erich Bruns, Benjamnin Brombach, Thomas Zeidler, and Oliver Bimber. Enabling mobile phones to support large-scale museum guidance. *IEEE Multimedia*, 14:16–25, 2007.

[19] Scott Buffett and Liqiang Geng. Bayesian classification of events for task labeling using workflow models. In *Business Process Management Workshops*, pages 97–108, Milano, Italy, September 2009.

[20] Chandra S. Chekuri, Andrew V. Goldberg, David R. Karger, Matthew S. Levine, and Cliff Stein. Experimental study of minimum cut algorithms. In *Proc. of the 8th annual ACM-SIAM symposium on Discrete algorithms*, SODA '97, pages 324–333, Philadelphia, PA, USA, 1997. Society for Industrial and Applied Mathematics.

[21] C. Cheng, Ravi Jain, and Eric van den Berg. *Handbook of Wireless Internet*, chapter Location prediction algorithms for mobile wireless systems. CRC Press, 2003.

[22] Carolina Chiao, Cirano Iochpe, Lucinéia Heloisa Thom, and Manfred Reichert. Verifying existence, completeness and sequences of semantic process patterns in real workflow processes. In *Proc. of the Simpsio Brasileiro de Sistemas de Informao. Rio de Janeiro: UNIRIO*, pages p. 164–175., Brazil, 2008.

[23] Tanzeem Choudhury, Matthai Philipose, Danny Wyatt, and Jonathan Lester. Towards activity databases: Using sensors and statistical models to summarize people's lives. *IEEE Data Eng. Bull.*, 29(1):49–58, 2006.

[24] E. Cuervo, A. Balasubramanian, D. Cho, A. Wolman, S. Saroiu, R. Chandra, and P. Bahl. Maui: Making smartphones last longer with code offload. In *Proc. of the 8th International Conference on Mobile systems, Applications, and Services (MobiSys)*, 2010.

[25] Brian D. Davison and Haym Hirsh. Predicting sequences of user actions. In *Workshop on Predicting the Future: AI Approaches to Time Series Analysis*, 1998.

[26] H. Falaki, R. Mahajan, S. Kandula, D. Lymberopoulos, R. Govindan, and D. Estrin. Diversity in smartphone usage. In *Proc. of the 8th international conference on Mobile systems, applications, and services (MobiSys)*, pages 179–194. ACM, 2010.

[27] Daniel Fischer, Stefan Föll, Klaus Herrmann, and Kurt Rothermel. Energy-efficient Workflow Distribution. In *Proceedings of The Fifth International Conference on COMmunication System softWAre and middlewaRE*, 2011. (accepted for publication).

[28] Stefan Föll, Klaus Herrmann, and Christian Hiesinger. Flow-Based Context Prediction. In *Proceedings of the 7th International Conference on Pervasive Services (ICPS 2010), Berlin, Germany, July 13-15, 2010*. ACM, Juli 2010.

[29] Stefan Föll, Klaus Herrmann, and Kurt Rothermel. Precon expressive context prediction using stochastic model checking. In *Proceedings of the 8th International Conference on Ubiquitous Intelligence and Computing (UIC-2011)*, 2011. (accepted for publication).

[30] Dimitra Giannakopoulou and Klaus Havelund. Automata-based verification of temporal properties on running programs. In *In Proceedings, International Conference on Automated Software Engineering (ASE01)*, pages 412–416. IEEE Computer Society, 2001.

[31] Karthik Gopalratnam and Diane J. Cook. Online sequential prediction via incremental parsing: The active lezi algorithm. *IEEE Intelligent Systems*, 22:52–58, 2007.

[32] S. Hariri H. Topcuoglu and W. Min You. Performance-effective and low-complexity task scheduling for heterogeneous computing. *IEEE Transactions on Parallel Distributed Systems*, 13:260–274, 2002.

[33] Gregory Hackmann, Christopher Gill, and Gruia-Catalin Roman. Extending bpel for interoperable pervasive computing. In *IEEE International Conference on Pervasive Services*, pages 204 – 213, Istanbul, 15-20 July 2007.

[34] Gregory Hackmann, Mart Haitjema, Christopher D. Gill, and Gruia-Catalin Roman. Sliver: A BPEL Workflow Process Execution Engine for Mobile Devices. In *Proceedings of 4th International Conference on Service Oriented Computing (ICSOC*, volume 4294 of *LNCS*, pages 503–508. Springer, 2006.

[35] Mark Hall, Eibe Frank, Geoffrey Holmes, Bernhard Pfahringer, Peter Reutemann, and Ian H. Witten. The weka data mining software: an update. *SIGKDD Explor. Newsl.*, 11(1):10–18, 2009.

[36] M. Hartmann and D. Schreiber. Prediction algorithms for user actions. In *In Proc. of Intl. Conf. on Adaptive Business Information Systems*, 2007.

[37] Klaus Herrmann, Kurt Rothermel, Gerd Kortuem, and Naranker Dulay. Adaptable Pervasive Flows–An Emerging Technology for Pervasive Adaptation. In *Proceedings of the 2008 Second IEEE International Conference on Self-Adaptive and Self-Organizing Systems Workshops*, pages 108–113. IEEE Computer Society, 2008.

[38] Christian Hiesinger, Daniel Fischer, Stefan Foell, Klaus Herrmann, and Kurt Rothermel. Minimizing Human Interaction Time in Workflows. In *Proceedings of the Sixth International Conference on Internet and Web Applications and Services (ICIW 2011)*, pages 22–28, March 2011.

[39] Ronald A. Howard. *Dynamic Probabilistic Systems: Semi-Markov and Decision Processes*. John Wiley & Sons, 1971.

[40] IBM. Web services business process execution language version 2.0.

[41] Java Universal Network/Graph. *http://jung.sourceforge.net*, 2010.

[42] David R. Karger and Clifford Stein. A new approach to the minimum cut problem. *J. ACM*, 43:601–640, July 1996.

[43] Dimitrios Katsaros and Yannis Manolopoulos. Prediction in wireless networks by markov chains. *IEEE Wireless Communications*, 16:56–63, 2009.

[44] Gerald G. Koch, Boris Koldehofe, and Kurt Rothermel. Higher confidence in event correlation using uncertainty restrictions. In *28th IEEE International Conference on Distributed Computing Systems Workshops (ICDCSW'08); 2008*, pages 417–422, 2008.

[45] Y. Kopidakis. On the task assignment problem: two new efficient heuristic algorithms. *Journal of Parallel and Distributed Computing*, 42(2):21, 1997.

[46] M. Kwiatkowska, G. Norman, and D. Parker. Stochastic model checking. *Formal Methods for the Design of Computer, Communication and Software Systems: Performance Evaluation*, 4486:220–270, 2007.

[47] Ralph Lange, Harald Weinschrott, Lars Geiger, Andre Blessing, Frank Dürr, Kurt Rothermel, and Hinrich Schütze. On a generic uncertainty model for position information. In Kurt Rothermel, Dieter Fritsch, Wolfgang Blochinger, and Frank Dürr, editors, *First Internationa Workshop on Quality of Context, QuaCon 2009*, number 5786 in LNCS, pages 76–87, Stuttgart, June 2009. Springer.

[48] Jean Michel Lau, Cirano Iochpe, Lucinéia Heloisa Thom, and Manfred Reichert. Discovery and analysis of activity pattern co-occurrences in business process models. In *ICEIS (3)*, pages 83–88, 2009.

[49] Jong-Known Lee and Jennifer C. Hou. Modeling steady-state and transient behaviours of user mobility: Formulation, analysis, and application. In *Proc. of the 7th ACM International Symposium on Mobile Ad Hoc Networking and Computing*, 2006.

[50] Frank Leymann and Dieter Roller. *Production workflow: concepts and techniques*. Prentice Hall PTR, 2000.

[51] Gabriel G. Infante Lopez, Holger Hermanns, and Joost-Pieter Katoen. Beyond memoryless distributions: Model checking semi-markov chains. In *Process Algebra and Probabilistic Methods. Performance Modeling and Verification*, 2001.

[52] Rene Mayrhofer. Context prediction based on context histories: Expected benefits, issues and current state-of-the-art. In *Proc. of the 1st Intl. Workshop on Exploiting Context Histories in Smart Environments*, 2005.

[53] Rene Michael Mayrhofer. *An Architecture for Context Prediction*. PhD thesis, Johannes Kepler University of Linz, 2004.

[54] Alan Messer, Ira Greenberg, Philippe Bernadat, Dejan Milojicic, Deqing Chen, T. J. Giuli, and Xiaohui Gu. Towards a distributed platform for resource-constrained devices. In *Proc. of the 22nd International Conference on Distributed Computing Systems (ICDS)*, 2002.

[55] Kevin Patrick Murphy. *Dynamic Bayesian Networks: Representation, Inference and Learning*. PhD thesis, UNIVERSITY OF CALIFORNIA, BERKELEY, 2002.

[56] B. Najafi, K. Aminian, A. Paraschiv-Ionescu, F. Loew, C.J. Bula, and P. Robert. Ambulatory system for human motion analysis using a kinematic sensor: monitoring of daily physical activity in the elderly. *Biomedical Engineering, IEEE Transactions on*, 50(6):711 –723, 2003.

[57] Brenda Ng, Leonid Peshkin, and Avi Pfeffer. Factored particles for scalable monitoring. In *In Proceedings of the Eighteenth Conference on Uncertainty in Artificial Intelligence*, pages 370–377. Morgan Kaufmann, 2002.

[58] M. Martin P. Nurmi and J. A. Flanagan. Enabling proactiveness through context prediction. In *In Proc. of the 2nd Workshop on Context Awareness for Proactive Systems*, 2005.

[59] Witold Pedrycz. Why triangular membership functions? *Fuzzy Sets and Systems*, 64:21 – 30, 1994.

[60] Witold Pedrycz and Fernando Gomide. A generalized fuzzy petri net model. *IEEE Transactions on Fuzzy Systems*, 2(4):295 –301, November 1994.

[61] Maja Pesic, Helen Schonenberg, and Wil M.P. van der Aalst. Declare: Full support for loosely-structured processes. *Enterprise Distributed Object Computing Conference, IEEE International*, 0:287, 2007.

[62] Lawrence R. Rabiner. A tutorial on hidden markov models and selected applications in speech recognition. *Proceedings of the IEEE*, 77:257–286, 1989.

[63] Andrei Radulescu and Arjan J. C. Van Gemund. Fast and effective task scheduling in heterogeneous systems. In *HCW '00: Proceedings of the 9th Heterogeneous Computing Workshop*, page 229, Washington, DC, USA, 2000. IEEE Computer Society.

[64] A.B. Raposo, A.L.V. Coelho, L.P. Magalhaes, and I.L.M. Ricarte. Using fuzzy petri nets to coordinate collaborative activities. In *IFSA World Congress and 20th NAFIPS International Conference, 2001. Joint 9th*, volume 3, pages 1494 –1499 vol.3, 2001.

[65] Stuart J. Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*. Prentice Hall, 2nd edition edition, 2002.

[66] Robert Schalkoff. *Pattern Recognition. Statistical, Structural and Neural Approaches*. John Wiley & Sons, Inc, 1992.

[67] Stephan Schuhmann, Klaus Herrmann, and Kurt Rothermel. A Framework for Adapting the Distribution of Automatic Application Configuration. In *Proceedings of the 2008 ACM International Conference on Pervasive Services (ICPS 2008), Sorrento, Italy, July 6-10, 2008*, pages 163–172. ACM, Juli 2008.

[68] Stephan Sigg. *Development of a novel context prediction algorithm and analysis of context prediction schemes*. PhD thesis, University of Kassel, 2008.

[69] Jin Hyun Son, Seok Kyun Oh, Kyung Hoon Choi, Yoon Joon Lee, and Myoung Ho Kim. GM-WTA: an efficient workflow task allocation method in a distributed execution environment. *Journal of Systems and Software*, 67(3):165–179, 2003.

[70] Libo Song, David Kotz, Ravi Jain, and Xiaoning He. Evaluating next-cell predictors with extensive wi-fi mobility data. *IEEE Transactions on Mobile Computing*, 5:1633–1649, 2004.

[71] Stephan. Urbanski, Eduard. Huber, Matthias. Wieland, Frank. Leymann, and Daniela. Nicklas. Perflows for the computers of the 21st century. In *Pervasive Computing and Communications, 2009. PerCom 2009. IEEE International Conference on*, pages 1 –6, March 2009.

[72] Wil M.P. van der Aalst, K.M. van Hee, and G.J. Houben. Modelling and analysing workflow using a petri-net based approach. In *Proc. 2nd Workshop on Computer-Supported Cooperative Work Petri nets and related formalisms*, pages pp 31–50, 1994.

[73] Mark Weiser. The computer for the 21st century. In *Scientific American 265(3): 94-104*, 1991.

[74] Mark Weiser. The computer for the 21st century. *SIGMOBILE Mob. Comput. Commun. Rev.*, 3(3):3–11, 1999.

[75] Matthias Wieland, Uwe-Philipp Käppeler, Paul Levi, Frank Leymann, and Daniela Nicklas. Towards Integration of Uncertain Sensor Data into Context-aware Workflows. In GI-Edition Lecture Notes in Informatics (LNI), editor, *Tagungsband INFORMATIK 2009 Im Focus das Leben, 39. Jahrestagung der Gesellschaft für Informatik e.V. (GI)*, Lübeck, September 2009. Lecture Notes in Informatics (LNI).

[76] Matthias Wieland, Oliver Kopp, Daniela Nicklas, and Frank Leymann. Towards context-aware workflows. In Barbara Pernici and Jon Atle Gulla, editors, *CAiSE07 Proceedings of the Workshops and Doctoral Consortium*, volume 2, Trondheim Norway, Juni 2007. Tapir Acasemic Press.

[77] Hannes Wolf, Klaus Herrmann, and Jonas Palauro. Fuzzy Event Assignment for Robust Context-Aware Workflows. In *Proceedings of the Fourth International Conference on Dependability (DEPEND 2011)*, 2011. (accepted for publication).

[78] Hannes Wolf, Klaus Herrmann, and Kurt Rothermel. Modeling dynamic context awareness for situated workflows. In P. Herrero R. Meersman and T. Dillon (Eds.), editors, *OTM 2009 Workshops*, volume 5872 of *LNCS*, pages 98–107, Vilamoura, November 2009. Springer-Verlag Berlin Heidelberg.

[79] Hannes Wolf, Klaus Herrmann, and Kurt Rothermel. Robustness in Context-Aware mobile computing. In *IEEE International Conference on Wireless and Mobile Computing, Networking and Communications (WiMob'2010)*. IEEE Communications Society, 10 2010.

[80] Hannes Wolf, Klaus Herrmann, and Kurt Rothermel. Robustness in Context-Aware mobile computing. In *IEEE International Conference on Wireless and Mobile Computing, Networking and Communications (WiMob'2010)*, Niagara Falls, Canada, 10 2010.

[81] LA Zadeh. Fuzzy sets. *Information and Control*, 8(3):338–353, 1965.

[82] Jacob Ziv and Abraham Lempel. Compression of individual sequences via variable-rate coding. *IEEE Transactions on Information Theory*, 24:530–536, 1978.