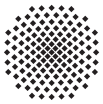

Universität Stuttgart

Reducing Context Uncertainty for Robust Pervasive Workflows

Von der Fakultät für Informatik, Elektrotechnik und Informationstechnik der Universität Stuttgart zur Erlangung der Würde eines Doktors der Naturwissenschaften (Dr. rer. nat.) genehmigte Abhandlung

Vorgelegt von

Hannes Wolf

aus Cottbus

Hauptberichter: Prof. Dr. rer. nat. Dr. h. c. Kurt Rothermel

Mitberichter: Prof. Dr. rer. nat. Paul Lukowicz

Tag der mündlichen Prüfung: 30.11.2015

Institut für Parallele und Verteilte Systeme (IPVS)

der Universität Stuttgart

2015

Acknowledgements

First of all, my most sincere gratitude belongs to my supervisor Prof. Dr. Kurt Rothermel, who provided me with the unique chance to be a member of the distributed computing research group at the University of Stuttgart. The time and effort he spent to keep my research on track and the encouraging and fruitful discussions, have been invaluable for the scientific contributions presented in this thesis. But apart from this personal guidance, I enjoyed being a member of his research group providing me a cooperative, helpful environment and a chance to broaden my view on computer networks and distributed systems.

In addition to that, I am very grateful towards Prof. Dr. Paul Lukowicz who accepted the role as second supervisor of my thesis. Moreover, he provided me with a very different, pragmatic and constructive view approaching my scientific challenges.

I also want to thank Dr. Klaus Herrmann for providing advice and guidance to develop the right set of skills required for research. Further, as a supervisor of my diploma thesis he paved the way for me joining academic research and made all this possible.

Special thanks go also to Stefan Föll, Christian Hiesinger, Agnes Grünerbl, Kai Kunze, Gernot Bahle, Tobias Unger, Hanna Eberle, Srdjan Marinovic and Bashar Altakrouri for good cooperation during the ALLOW research project. It has been a very valuable experience for me to work and research in an international team distributed all over Europe.

Finally, I would like to thank my wife and family for enduring the much to long time period while I slowly managed to get this thesis written. Your patience and support have been a significant contribution to make this process complete successfully, eventually.

Abstract

Mobile computing devices equipped with sensors are ubiquitously available, today. These platforms provide readings of a multitude of different sensor modalities with fairly high accuracy. But the lack of associated application knowledge restrains the possibility to combine this sensor information to accurate high-level context information. This information is required to drive the execution of applications, without the need for obtrusive explicit human interaction.

A modeled workflow as formal representation of a (business) process can provide structural information on the application. This is especially the case for processes that cover applications with rich human interaction. Processes in the health-care domain are characterized by coarsely predefined recurring procedures that are adapted flexibly by the personnel to suite specific situations and rich human interaction. In this setting, a workflow managment system that gives guidance and documents staff actions can lead to a higher quality of care, fewer mistakes, and a higher efficiency. However, most existing workflow managment systems enforce rigid inflexible workflows and rely on direct manual input. Both is inadequate for health-care processes.

The solution could be activity recognition systems that use sensor data (e. g. from smart phones) to infer the current activities by the personnel and provide input to a workflow (e.g. informing it that a certain activity is finished now). However, state of the art activity recognition technologies have difficulties in providing reliable information.

In this thesis we show that a workflow can aid as source of structural application knowledge for activity recognition and that the other way around, a workflow can be driven by context information in a way reducing the need for explicit interaction. We describe a comprehensive framework – FlowPal– tailored for flexible human-centric processes, that improves the reliability of activity recognition data. FlowPals set of mechanisms exploits the application knowledge encoded in workflows in two ways. ⋆Con (StarCon) increases the accuracy of high-level context events using information from an associated workflow. Fuzzy Event Assignment (FEvA) mitigates errors in sequences of recognized context. This way FlowPal enables unobtrusive robust workflows.

We evaluate our work based on a real-world case study situated in the health-care domain and show that the robustness of unobtrusive health-care workflows can be

increased. With ⋆Con we can improve the accuracy of recognized context events up to 56%. Further we enable the successful execution of flows for a uncertain context events large range of uncertain context events, where a reference system fails. Overall, we achieve an absolute flow completion rate of about 91% (compared to only 12% with a classical workflow system). Our experiments also show that FEvA achieves an event assignment accuracy of 78% to 97% and improves the performance of dealing with false positive, out-of-order events and missed context events.

Deutsche Zusammenfassung

Die letzten zwei Jahrzehnte waren geprägt durch die rasante Entwicklung des mobilen Internets und der allgegenwärtigen Verfügbarkeit von mobilen Smartphones. Dieser Entwicklung haben wir es zu verdanken, dass Menschen jederzeit und überall miteinander Informationen austauschen und kommunizieren können. Allerdings hat sich die Art und Weise, wie wir mit mobilen Endgeräten interagieren, weitaus langsamer entwickelt. Das Nutzungskonzept konzentriert sich auf das Anzeigen von Informationen auf einem Display und die direkte Eingabe via einer (virtuellen) Tastatur. Ausnahmen dazu sind das Aufnehmen von Bildern oder die teilweise verbreitete Anwendung von Spracheingabe. Dennoch werden diese Interaktionen alle unmittelbar durch den Nutzer ausgeführt und erfordern einen großen Teil seiner Aufmerksamkeit um die (mobilen) Anwendungen zu steuern.

Ein weiterer Trend, der bereits dabei ist, mobile Endgeräte deutlich zu verändern, ist die hohe Verfügbarkeit von immer besseren und immer kleineren Sensoren. Sie erlauben es nahezu jedem mobilen Endgerät Informationen aus seiner direkten Umgebung und über seinen Nutzer zu sammeln und auszuwerten. Heutige Sensoren sind in der Lage eine große Vielzahl von unterschiedlichen Informationen zu sammeln. Die geläufigste Informationsquelle ist die Position des Gerätes (bzw. Benutzers), die im Freien mit dem Global Positioning System (GPS) leicht ermittelt werden kann. Weiterhin gehören sowohl Kameras als auch Mikrophone zur Grundausstattung moderner mobiler Endgeräte. Zusätzlich zu ihren Hauptaufgaben, wie der Aufnahme von Bildern oder Videos inklusive Ton, können diese Sensoren auch dafür genutzt werden, Objekte oder Personen zu erkennen oder Anwendungen mit Gesten bzw. der Stimme zu steuern.

Moderne mikromechanische Sensoren erlauben das Erfassen von Beschleunigungen, die das Smartphone erfährt, und damit das Erkennen von Änderungen in der Bewegung, dsm freien Falls des Gerätes oder der Position relativ zum Boden. Eine einfache Anwendung für diese Daten ist das Drehen der Anzeige relativ zur Lage des Gerätes. Für weitergehende Anwendungen, die auf der absoluten Lage des mobilen Endgerätes im Raum angewiesen sind, stehen Drehraten und Magnetfeldsensoren zur Verfügung. Außerdem werden immer häufiger auch Drucksensoren verbaut, mit denen es möglich ist, die geographische Höhe auch anhand

des Luftdrucks zu ermitteln. All diese Sensoren erleichtern die Positionsbestimmung und Navigation des mobilen Gerätes sowohl im Freien wie auch in Gebäuden.

Aber werden die einzelnen Sensorinformationen nur isoliert betrachtet, geben sie nur ein beschränktes Bild auf die tatsächliche Situation des Gerätes oder Nutzers wieder. Ein wesentlich größerer Informationsgewinn ergibt sich, wenn man die Sensorinformationen miteinander kombiniert und auch noch implizites Wissen oder Anwendungswissen hinzunimmt. Auf diese Weise kann abstrakteres und komplexeres Wissen über den aktuellen Kontext gewonnen werden. Als Beispiel betrachten wir einen Krankenpfleger, der für einige seiner Tätigkeiten zusätzliche Informationen von einer Anwendung auf einem mobilen Endgerät benötigt. Damit die Anwendung ohne direkte Eingaben des Pflegers funktioniert, muss sie über durchgeführte Tätigkeiten des Pflegers per Kontexterkennung informiert sein. Dazu gehören zum Beispiel die richtige Ausstattung des Pflegers (Handschuhe), die gerade durchgeführte Tätigkeit (Medikamenteneinnahme kontrollieren) oder die Art und Weise wie diese Tätigkeit durchgeführt wurde. Es ist offensichtlich, dass der Pfleger diese Informationen nur schlecht eingeben kann, während er die Tätigkeit durchführt. Auch ist es kaum möglich, die nötigen Informationen vorab genau zur Verfügung zu stellen, da es notwendig werden kann, den Ablauf kurzfristig anzupassen. Daher kann eine solche Anwendung nur durch indirekte Interaktion mit dem Nutzer effektiv verwendet werden. Durch die Verwendung von abstrakten Informationen aus der Kontexterkennung können Anwendungen durch den Menschen effektiver und mit weniger direkter Interaktion genutzt werden.

Leider sind jedoch alle Sensorinformationen bei der Messung mit einer Messunsicherheit behaftet. Diese Messunsicherheit kann durch Genauigkeit und Präzision näher definiert werden. Die Genauigkeit gibt an, wie nah der aktuell gemessene Wert einen Referenzwert widerspiegelt. Die Präzision der Messung gibt die statistische Abweichung von diesem Referenzwert bei wiederholter Messung unter gleichen Bedingungen an. Je höher die Genauigkeit und Präzision einer Messung ist, desto weniger unsicher ist die ermittelte Information. Die Sensoren, die in modernen mobilen Endgeräten zur Verfügung stehen, haben eine weitgehend akzeptable Genauigkeit. Eine hohe Präzision kann durch hohe Abtastraten und die Mittlung von mehreren Einzelmessungen erreicht werden.

Damit ergibt sich für einzelne Messwerte eine vertretbare Messunsicherheit. Jedoch multiplizieren sich diese Unsicherheiten bei der Kombination von Sensorinformationen, so dass die Gesamtunsicherheit stark zunimmt.

Um über die Unsicherheit eines einzelnen abstrakten Ereignisses genauer argumentieren zu können, werden neben der Information das ein Ereignis stattgefunden hat auch Informationen über die Unsicherheit übertragen. Dazu ist ein geeignetes Unsicherheitsmodell erforderlich. Üblicherweise wird diese Unsicherheit

unter Verwendung von einfacher Wahrscheinlichkeitstheorie durchgeführt und auch die weitere Verrechnung von diesen Ereignissen geschieht unter Berücksichtigung dieser Wahrscheinlichkeiten. Doch je geringer die Signifikanz eines Ereignisses ist und je weiter es verarbeitet wird, desto schwerwiegender wirkt sich die Unsicherheit aus. Die abstrakten Ereignisse sind dann nur noch schlecht zu erkennen und potenziell mehrdeutig. Daher stellt die robuste Erkennung von abstrakten Nutzertätigkeiten immer noch eine herausfordernde Aufgabe dar. Weitere Gründe, die dieses Problem verdeutlichen, sind die Auswahl der geeigneten Sensoren, mangelndes Wissen über die Aktion aus der Anwendung, Probleme bei der geeigneten Platzierung von Sensoren und natürlich die Kosten [BTJ+10]. Unter diesen Bedingungen können verschiedene Fehler auftreten. Zunächst kann es zu sogenannten falsch positiven Ereignissen kommen, die zwar erkannt werden, aber nie wirklich stattgefunden haben. Eine zweite Art von Fehlern sind sogenannte falsch negative Ereignisse die passiert sind, aber nicht erkannt werden. Auch können erkannte Ereignisse aufgrund der Unsicherheit falsch interpretiert werden und daher ungewollte Reaktionen der Anwendung hervorrufen.

In einem gänzlich anderen Bereich gab es in den letzten Jahrzehnten einen ähnlich prägenden Trend. Mit der immer stärkeren Verbreitung von Computern haben Unternehmen begonnen, standardisierte Geschäftsfälle so weit wie möglich zu automatisieren. Diese Entwicklung setzte sich zunächst in den klassischen "Papierfabriken" durch. Typische Vertreter sind Banken, die regelmäßig standardisierte Anträge zur Kontoeröffnung durchführen, und Versicherungenm die Schadensmeldungen bearbeiten müssen. Diese Automatisierung von Geschäftsprozessen wurde schnell auch in anderen Branchen wie dem produzierenden Gewerbe übernommen [LR00]. Durch die konsequente Anwendung konnten die Unternehmen eine größere Anzahl von standardisierten Geschäftsfällen mit weniger Personal bewältigen.

Als Folge dieser Veränderung sind abstrakte Modellierungssprachen entwickelt worden, um Geschäftsprozesse formal zu beschreiben und diese Beschreibung soweit möglich maschinell mit einer Prozess-Engine auszuführen. Grundsätzlich definiert jede Prozessbeschreibungsprache Sprachelemente um einzelne Prozessschritte (Aktivitäten) zu beschreiben und ihren Ablauf anhand von logischen oder daten-basierten Abhängigkeiten zu steuern. Auf diese Weise ist es möglich eine große Zahl von Geschäftsfällen teil- oder vollautomatisch auszuführen.

Historisch bedingt sind die meisten dieser Modellierungssprachen jedoch für die Automatisierung von hoch standardisierten Abläufen geschaffen worden. Daher sind solche modellierten Prozesse in der Regel nur schlecht in der Lage, auf kurzfristige Änderungen im Geschäftsvorfall zu reagieren. Außerdem wird die Interaktion von Menschen mit automatisierten Geschäftsprozessen ebenfalls stark standardisiert und beschränkt sich typischerweise auf das Bereitstellen und Bewerten von Informationen sowie das Treffen von Entscheidungen. Die Entwicklung mo-

derner Prozessbeschreibungssprachen orientiert sich jedoch auch an komplexeren Arbeitsumgebungen und der damit notwendigen höheren Flexibilität beim Ausführen der Geschäftsprozesse. Beispiele hierfür sind Prozesse, die in Forschung und Entwicklung eingesetzt werden (vgl. [BH08]). Mit der Erweiterung der Anwendungsfälle sind jedoch auch die Anforderungen an Geschäftsprozesse gestiegen, direkte Tätigkeiten der menschlichen Anwender in komplexen Situationen zu unterstützen. Besonders is dies der Fall in Anwendungsgebieten bei denen Interaktion zwischen Menschen den Arbeitsalltag prägen. Ein Beispiel dafür ist der medizinische Bereich, in dem Ärzte und Pfleger am und mit dem Patienten arbeiten. Zeitgleich gibt es jedoch strukturierte Prozesse, die eingehalten werden müssen und umfangreiche Richtlinien, die zu beachten sind. Einerseits könnte ein automatischer Geschäftsprozess Unterstützung bei der Durchführung und auch beim Einhalten der Richtlinien geben. Andererseits muss er möglichst vollständig durch die Interaktion der Anwender (Ärzte, Pfleger) mit den Patienten bzw. geeigneten Werkzeugen gesteuert werden.

Im Rahmen des ALLOW Projekts [HRKD08] sind daher sogenannte "adaptive pervasive flows" (APF) als modernes Programmierparadigma vorgeschlagen und untersucht worden, um als Basis für die Modellierung von Anwendungen zu dienen, welche die oben genannten Anforderungen erfüllen können. Dabei ist ein APF ein Prozess der im Hintergrund läuft und die Anwender durch das Vorbereiten, Bereitstellen oder Anpassen von Ressourcen und Werkzeugen unterstützen kann (z.B. Konfigurieren einer Anwendung).

Einer der Gründe dafür, dass solche Prozesse heute noch nicht möglich sind, ist die erwähnte Unsicherheit bei der Erkennung von Kontextinformationen. Bisherige Versuche dieses Problem zu lösen, hatten nur beschränkten Erfolg, weil beim Erkennen des richtigen Kontextes zu wenig strukturiertes Wissen über die Anwendung bzw. das Anwendungsumfeld vorhanden waren [BTJ+10]. Dieses Anwendungswissen steuert wichtige Informationen über die gerade durchgeführten Aktivitäten bei. Daher kann dieses Wissen dafür genutzt werden, die Probleme bei der Kontexterkennung zu reduzieren und bei der richtigen Interpretation von unsicheren Informationen über den Nutzerkontext zu helfen.

Im Rahmen dieser wissenschaftlichen Ausarbeitung ist das Kernthema die Erforschung des Zusammenspiels zwischen automatisch ablaufenden, kontextgetriebenen Geschäftsprozessen und der Erkennung von komplexen menschlichen Tätigkeiten mit unsicheren Sensorinformationen auf (mobilen) Endgeräten. Dabei stehen zwei Arten der Interaktion zwischen Prozess und Kontext im Fokus. Zum Einen steuert der menschliche Anwender den Geschäftsprozess weitgehend, ohne dass der Anwender aktiv mit dem Prozess interagieren muss. Zum Anderen profitiert die Erkennung der menschlichen Tätigkeiten deutlich durch die gespeicherte Prozesslogik und die Anwendungsstruktur.

Ganz konkret wird untersucht, auf welche Weise die Struktur des modellierten Prozesses genutzt werden kann, um die Unsicherheit und Mehrdeutigkeit beim Erkennen von abstrakten Kontextinformationen zu reduzieren. Diese werden wiederum gebraucht, um die Anwendung stabil auszuführen. Die Untersuchung dieser Fragestellung führte zu den folgenden konkreten Beiträgen.

Zunächst wurde das (1) "Hybrid Flow Model" (HFM) [WHR11] entwickelt, welches eine Prozessbeschreibungssprache darstellt. Das HFM erlaubt die Verwendung der beiden gängigen Prozessmodellierungskonzepte (imperativ und deklarativ) im selben Prozessmodell und damit auch auf derselben Abstraktionsebene. Mit dem HFM kann man einerseits stark strukturierte Prozesse modellieren, wie sie in klassischen Anwendungsszenarien vorkommen. Dabei werden die Aktivitäten anhand von festen Transitionen mit geeigneten Kontrollbedingungen geordnet. Andererseits unterstützt es auch die Beschreibung von Prozessen mit lose gekoppelten Aktivitäten, deren Struktur nur über deklarative Bedingungen ("Constraints") vorgegeben sind. Dabei sind deklarative Prozesse besser geeignet, um Menschen bei ihren Aufgaben zu unterstützen, da sie lediglich vorgeben was getan werden muss, aber nicht genau wie und wann. Weiterhin ist das HFM dafür ausgelegt, Anwendungen zu modellieren die hauptsächlich durch Kontextereignisse gesteuert werden. Dabei stellt die unmittelbare Kombination der Modellierungskonzepte und die damit einhergehende höhere Flexibilität beim Modellieren und Ausführen einen neuen Beitrag dar.

Basierend auf dem HFM wurden eine Reihe unterschiedlicher Mechanismen entwickelt, welche Informationen aus der Struktur eines Prozesses nutzen. Diese Mechanismen sind in der (2) "FlowPal"-Architektur [WHR13] zusammengefasst. Sie bietet den Rahmen, um die verschiedenen Mechanismen zu bündeln und ihre gemeinsame Funktionsweise und gegenseitige Unterstützung darzustellen. Außerdem können durch "FlowPal" die Methoden flexibel als zusätzliche Komponenten einer Prozess-Engine eingesetzt werden.

Der erste entwickelte Mechanismus ist das (3) "Flow Context System" (FlowCon). FlowCon [WHR10] ist ein Plug-In für eine Prozess-Engine, das speziell für die klassischen hoch-strukturierten Prozesse ausgelegt ist. Die Anwendung von "FlowCon" ist insbesondere dann sinnvoll, wenn die Ausführung der Prozesse stark durch Kontextinformationen getrieben wird. "FlowCon" nutzt dabei die vorhandenen strukturellen Informationen aus einem Geschäftsprozess, um statistische Abhängigkeiten der Kontextinformationen zu ermitteln, die für die Ausführung des Geschäftsprozesses relevant sind. Dazu geht "FlowCon" zweistufig vor. Zunächst werden in einem ersten Schritt die Transitionen des Prozessmodels und die zugehörigen Kontextinformationen analysiert. Mit diesen Informationen wird dann die Struktur eines Bayeschen Netzwerkes aufgebaut, welches die bedingten Abhängigkeiten der Kontextinformation anhand des Prozessmodels wiedergeben kann. Aus historischen Daten über bereits erfolgreich ausgeführte Instanzen

des Geschäftsprozesses können dann im zweiten Schritt die Gewichte im Bayeschen Netzwerk angepasst werden. Diese entsprechen dann den bedingten relativen Häufigkeiten der Kontextinformationen wie sie in den historischen Daten vorkamen. Das Netzwerk kann nun aber auch, unter Verwendung eines aktuellen Ausführungszustandes des Geschäftsprozesses, dafür genutzt werden, die aktuellen vorliegenden Kontextinformationen hinsichtlich ihrer statistisch historischen Häufigkeit zu bewerten. Mit dieser Bewertung kann die Interpretation der Kontextinformationen unmittelbar beeinflusst werden. In Experimenten wurde nachgewiesen, dass "FlowCon" in der Lage ist unter bestimmten Bedingungen die Genauigkeit von Kontextinformationen um bis zu 50% gegenüber der Ausgangsgenauigkeit zu steigern. Auf diese Weise kann die robuste Verarbeitung von unsicheren Kontextinformationen deutlich gesteigert werden. Außerdem war es möglich, die Rate der erfolgreich ausgeführten Prozesse unter diesen Bedingungen signifikant zu steigern.

Das Konzept von FlowCon ist anschließend erweitert und angepasst worden, um auch auf den flexibleren Geschäftsprozessen des "HFM" vollständig anwendbar zu sein. Das Ergebnis ist (4) "FlexCon" [WHR11]. Da das grundsätzliche Prinzip von "FlexCon" ist mit dem von "FlowCon" vergleichbar ist, werden beide auch als "⋆Con" (StarCon) bezeichnet. Auch bei "FlexCon" wird das Model des Geschäftsprozesses analysiert. Allerdings müssen nun ebenfalls aus den Constraints entsprechende Abhängigkeiten zu den Kontextinformationen gewonnen werden. Außerdem kann die bisherige statische Struktur des Bayeschen Netzwerks den dynamischen Ablauf des Geschäftsprozesses nicht mehr richtig abbilden. Daher muss auch für das statistische Schließen auf Dynamische Bayesche Netzwerke als flexibleres Werkzeug zurückgegriffen werden. Mit diesen Anpassungen ist "FlexCon" fähig eine ähnliche Verbesserung wie "FlowCon" auch für Geschäftsprozesse auf Basis des HFM zu liefern. Die Experimente zeigen hier eine Steigerung der Genauigkeit von Kontextinformationen um bis zu 73%. Gleichzeitig kann auch die Rate der korrekt ausgeführten Prozesse im Vergleich zum Referenzsystem wieder gesteigert werden. Allerdings fällt dieser Sprung wesentlich kleiner aus, da es nicht in allen Fällen möglich ist zusätliche Informationen aus "FlexCon" zu gewinnen.

Beide Mechanismen, "FlowCon" und "FlexCon" wurden weiter verbessert. Neben klassischer Wahrscheinlichkeitstheorie wurden dazu auch fortgeschrittenere Unsicherheitsmodelle untersucht, um die unsicheren Kontextinformationen darzustellen und zu kombinieren. Zum einen wurde mit Subjektiver Logik ein Modell untersucht, dass die explizite Darstellung von Unsicherheit in zweiter Ordnung unterstützt und eine vollständige Arithmetik für den Umgang mit unsicheren Aussagen bietet. Basierend auf subjektiver Logik lässt sich auch der Umfang genauer steuern den "StarCon" auf Kontextereignisse im konkreten Fall ausüben soll. Weiterhin wurde Vierwertige Logik verwendet, um bei der Auswertung von Bedingungen an den Transitionen auch explizit widersprüchliche Informationen

zu verwenden und zur Auflösung von Unsicherheit zu nutzen. Das Ergebnis dieser Untersuchungen ist (5) "HyperFlowCon" [WHR13] welches mit der Anwendung von subjektiver Logik und einigen weiteren Optimierungen noch einmal signifikante Verbesserungen bis hin zu 97% hinsichtlich der Rate der korrekt ausgeführten Prozesse erreicht. Außerdem kann die Sicherheit mit der die Prozess-Engine Entscheidungen trifft durch die Anwendung von "HyperFlowCon" ebenfalls deutlich gesteigert werden.

Die bisher vorgestellten Methoden zielen alle auf die Verbesserung der Genauigkeit eines einzelnen Ereignisses im Zusammenhang bei der Prozessausführung. Im Gegensatz dazu wurde mit dem (6) "Fuzzy Event Assignment" (FevA) [WHP11] einen Mechanismus entwickelt, der Fehler in einer Sequenz von Kontextereignissen, die beim Ausführen eines Geschäftsprozesses auftreten können, erkennen und beheben kann. Dazu erweitert "FEvA" den Zustandsraum der Aktivitäten, um rechtzeitig über Kontextinformation für zukünftige Aktivitäten informiert zu sein. Außerdem werden die Ereignisse in einem sogenannten Ereignis-Container zwischengespeichert, der dann mit Hilfe von zwei entwickelten Algorithmen die Kontextereignisse der richtigen Aktivität zuordnen kann. Dazu wird zunächst durch den "Candidate Selection Algorithm" mittels Fuzzy Logik eine linguistische Abbildung der Kontextereignisse auf die möglichen Aktivitäten errechnet. In einem zweiten Schritt bildet der "Event Assignment Algorithm" die Ereignisse mit der besten Entsprechung auf die richtige Aktivität ab. Dabei ist "FEvA" in der Lage sich adaptiv auf das Eintreffen neuer Ereignisse anzupassen und die Zuordnung entsprechend zu optimieren. Die durchgeführten Versuche zeigen, das "FEvA" die Ereignisse in bis zu 94% der Fälle richtig zuordnen kann. Für falsch positive Ereignisse liegen die Werte darunter aber immer noch um 16 bis 30 Prozentpunkte über dem Referenzsystem.

Die bisher genannten Ergebnisse wurden alle im Rahmen einer zweistufigen Evaluationsmethode gewonnen. Zunächst wurde eine intensive (7) Szenarioanalyse im medizinischen Pflegebereich durchgeführt [KBNL11]. Ein Ergebnis davon waren konkrete Anwendungsfälle und Strukturen für Prozesses die den formulierten Anforderungen an die Anwendung genügen. Auf Basis dieser Szenarioanalyse wurde eine Studie entwickelt, bei der auch umfangreiche Kontext-Informationen erhoben und ausgewertet wurden. Diese Daten lieferten wiederum wertvolle Impulse für die Gestaltung der weiteren Experimente. Um die entwickelten Mechanismen an einer hinreichend großen Anzahl von Geschäftsprozessmodellen zu testen, musste die Datenbasis weit über die gewonnenen Informationen aus der Studie erweitert werden. Dazu wurden Informationen aus anderen Studien, sowie über sogenannte "Workflow-Activity Patterns" [CITR08] verwendet um maschinell ähnliche Geschäftsprozesse zu erzeugen [WHR11]. Diese konnten dann als Basis für umfangreiche Simulationen dienen. Weiterhin konnten die gesammelten Kontextinformationen genutzt werden, um die Simulation mit praxisnahen Daten über die Kontexterkennung zu versorgen.

Diese Beiträge und die gewonnen Erkenntnisse stellen einen deutlichen Wissensngewinn im Bereich der kontextbezogenen prozessbasierten Anwendungen dar.

Contents

List of Figures

List of Tables

List of Algorithms

List of Acronyms and Symbols

The following acronyms are used throughout this document:

4V four-valued logic

APF Adaptable Pervasive Flow

bba basic belief assignment

BN Bayesian Network

BPEL Business Process Execution Language

BPM business process management

BPMN Business Process Modeling Notation

CE Condition Evaluator

CMS Context Management System

CPT conditional probability table

DS Dempster-Shafer Theory of Evidence

DBN Dynamic Bayesian Network

FEvA Fuzzy Event Assignment

FlexCon Flexible Flow Context (System)

FlowCon Flow Context (System)

HyperFlowCon Hyper Flow Context (System)

HFM Hybrid Flow Model

PN Petri Net

PT Probability Theory

RV random variable

SOA Service Oriented Architecture

SL Subjective Logic

SLCE Subjective Logic Condition Evaluator

WS Web Service Stack

WSDL Web Service Definition Language

YAWL Yet Another Workflow Language

List of Symbols

α	fraction of false positive events in an event sequence
A	a set of activities
a	an activity
c	a transition condition
C	a set of transition conditions
γ	fraction of out-of-order events in an event sequence
D	set of dependencies in a (dynamic) Bayesian Network
δ	fraction of deleted events in an event sequence
e	an instance of a context event
E	an event type set
I_E^e	interpretation of an context event instance e of event type set E
κ	fuzzy activity weighting function
l	a flow constraint
L	a set of constraints
λ	fuzzy event type weighting function
N	a set of Nodes (random variables) in a Bayesian Network, the joint probability distribution is described by the Bayesian Network
M	a set of marked transitions
μ	fuzzy membership function
$\mathcal{PD}$	prior distribution of a dynamic Bayesian Network
$\mathcal{S}$	a sequence of context Events
t	a transition
T	a set of Transitions
$\mathcal{T}$	a flow trace
$\mathcal{TM}$	transition model of a dynamic Bayesian Network
Θ	a frame of discernment/universe of events
u	a certain event type
U	the univers of all possible event types
ϕ	a function selecting mandatory activities
Φ	the set of all mandatory activities in a flow
X	discrete random variable
$\bar{X}$	set of random variables, their joint probability distribution can be represented in a (dynamic) Bayesian Network

$\bar{X}_t$	random variable X defined for time slice t in a dynamic Bayesian Network
χ	a function for mapping any activity - event type set pair to a single random variable in a Dynamic Bayesian Network, part of the FlexCon Dynamic Bayesian Network construction algorithm
Z	set containing all activity states
ω	activity state function

Part I.

Introduction, Background and System Model

1. Introduction

> The most profound technologies are those that disappear. They weave themselves into the fabric of everyday life until they are indistinguishable from it.
>
> Mark Weiser [Wei91]

1.1. Motivation

One of the most distinct success stories of the last decade has been the rapid development and broad availability of mobile computing systems. Devices like smart mobile phones or tablets have become ubiquitous and allow humans to communicate and share information almost any-time and everywhere in ways rarely imaginable ten years ago. While this has greatly influenced when and where people use information technology, the means of interaction with mobile computing systems have adapted much slower. Interaction is mostly focused on viewing information on some portable display and entering commands using a keyboard, touch screens or even voice activation. This interaction is explicitly driven by the users and they have to control applications directly.

There is another trend ongoing that is on its way to shape the next decade of information systems and interaction with them. Sensors are becoming an integral part of basically every smart mobile device, making the device aware to its surroundings and also to the users current context. A multitude of different modalities can be sensed today, already . The most common context information are position information that can be acquired outdoors using the global positioning system (gps). Furthermore cameras and microphones are standard sensing devices in smart mobile devices. In addition to their obvious usage to take pictures, create videos and record and transmit audio, they can be used to identify objects and people, or to control the device by gestures or voice commands. Also, features like noise cancellation and background noise detection require a microphone sensor. Next, there are sensors to measure acceleration of the mobile device, which can be used to detect changes in movement, free fall or position of the device relative to the ground. A simple application for acceleration data is the change of display orientation. For

more advanced movement detection, which enables full inertial navigation, additional gyroscope sensors are available that detect yaw rates applied to a device. Magnetic field sensors are used for compass applications and can in general be used to detect changes in the magnetic surroundings of the device. A relatively new type of sensors for mobile devices are pressure sensors, which can be used to estimate the barometric height, aiding both indoor and outdoor navigation solutions.

Being used on their own the sensors provide only a limited view on the device and user context. The range of possible applications becomes much larger as more of these sensor modalities are combined and used to cross check the plausibility of sensor readings. This way, much more complex and abstract high-level context information can be derived. For example, imagine a maintenance worker, that wants additional information for some of his tasks. In order to guide the maintenance worker an application needs events such as, getting the correct tool, using that tool for a specific action and how this action has been performed. If the maintenance worker has to enter each step on the fly, or has to describe all the steps before the actual task, the usability of the application would be low. Relying on explicit user interaction, prevents such an application to be effectively used. Using high-level abstract context information to guide the human user, allows for more expressive and less obtrusive interaction (cf. [KWK+09]).

Sensor data is always uncertain, but can be quantified by accuracy and precision. While accuracy defines how close the current reading represents the actual measured value, precision denotes the statistical deviation when the measurement is repeated under unchanged conditions. The more accurate and precise the measurement is the less uncertain the measured information is. The sensors used in commodity smart mobile devices have a pretty fair accuracy and a good precision can be achieved by averaging between multiple measurements. Hence, raw sensor data represents a valid measurement. But the more the data is processed to recognize high-level abstract context information, the more assumptions have to be included. Typically, those recognition results suffer from added uncertainty and ambiguity. Robustly detecting human activities in the real-world is still a very challenging task. The reasons are the difficulty of selecting appropriate sensors, the lack of application knowledge, the difficulty of sensor deployment, and low cost of deployed components [BTJ+10]. This leads to different types of errors like false positives (activities that are detected even though they never happened), false negatives (activities that remain undetected), or misclassifications (activities that actually happened but were assigned a false meaning).

In a very different area of computer science another trend has been ongoing for about twenty years. Starting in the early nineties, companies began to automate the execution of standard cases in their businesses processes in a large scale. First, this happened in typical "paperwork factories" like insurance companies and fi-

nancial industry for processes such as processing of damage reports and credit applications. They have been quickly adopted for manufacturing related automation of business processes [LR00]. Automating their *workflows* companies aimed for higher throughput of cases and a lower human resource consumption in workplace environments for repetitive tasks.

Abstract languages to describe such a workflow have been developed. Basically, a workflow language defines modeling elements to describe an structured application based on a set of tasks that are executed due to some additional logical and data dependencies. Due to their origin, most workflow languages are tailored to automate business processes, which are standardized and have low flexibility but a large number of cases to be processed. Human interaction with those type of workflows is usually limited, to entering or editing documents and data as well as decision making and approval.

As business process automation tried to capture more complex workplace environments, the workflows had to deal with greater flexibility in their execution. Nowadays, workflows are also used to coordinate creative engineering processes as well as scientific research. (c.f. [BH08])

But as the application areas of workflows increase, the requirements to integrate the human real-world interaction tightly into their execution becomes more challenging. This is especially the case in domains with rich human interaction, such as health-care [DRK00] (e. g. patient treatment, surgery, hospital care) or service industries (e. g. repair shops, logistics support). In this areas humans directly interact with each other or with goods and by their interaction drive the execution of a business process.

In the ALLOW project, we have proposed the so-called Adaptable Pervasive Flow (APF) (in short: flow) [HRKD08]. A flow is a programming paradigm to model real-world processes executed by humans. Running in the background a flow can document the work process for legal reasons or for subsequent analysis and optimization. It can provide guidance when users deviate from its definition. Further, a flow can support users by preparing, providing, or adapting resources (e. g. setting up required applications in a computer, or controlling applications for the user).

One of the reasons that all of this is not already a reality today is the aforementioned unreliability of current activity recognition technology. Attempts tackling this problem had limited success due to little or no structured knowledge about the application or its domain. Structured application knowledge provides semantic information about the activities in the real world. Therefore, this source of information can help to identify the described errors and interpret uncertain context information correctly.

1.2. Contributions

The key topic of this thesis is to explore the space of interaction between a flow, as structured workflow-like application that relies on tight interaction with a humans real-world actions, and the recognition process of high-level abstract context information from available sensors in mobile computing systems. We focus on two different ways of interaction between workflows and context information. First, a workflow that is enacted by a human should be automated solely using context information without obstructing the human in his behavior. Second, the recognition of uncertain context information benefits from information about business logic and application structure encoded in workflows.

More specifically we investigate in which ways structural information encoded in a flow can be used to reduce uncertainty and ambiguity in the context information that are required by the application. The research of this interaction space has led to the following individual contributions.

We have developed our *Hybrid Flow Model (HFM)* (1) [WHR11] for describing process-based applications. We investigated existing workflow languages and the modeling paradigms used for creating process-based applications. The Hybrid Flow Model combines the two widely used modeling paradigms, imperative(e. g. [AH03]) and declarative[PSA07]. On the one hand, it allows to model highly structured workflows that have been used for the classical applications such as those in insurance companies or manufacturing that we mentioned earlier. On the other hand, the HFM supports the creation of workflows using declarative expressions. These are much more suited supporting a human in his daily routine, as they mostly describe what to do and not exactly how and when. Using the HFM we can cover both highly structured and human-centric declarative flows. It is also a generic model that we use as foundation to build further process-context interaction mechanisms.

While researching the interaction space of context information and workflows we developed a number of mechanisms. All of them have been classified and subsumed under a common system architecture. This overall system architecture is *FlowPal* (2) [WHR13]. It provides us with the necessary framework to plug-in each of the mechanisms and show how they work together and complement each others function.

The *Flow Context (System) (FlowCon)* (3) [WHR10] is the first mechanism. It is a flow engine plug-in that is tailored to classical highly structured workflows that rely on context information for their successful execution. FlowCon extracts information from the flow structure to derive dependencies between context information which drive the flow execution. The gained knowledge is used to train a Bayesian Network (BN). The information stored in the BN is allows FlowCon to

increase the recognition accuracy of the related human activities, thus decreasing their uncertainty. This way the flow execution becomes more robust. That means, even when the original accuracy of the available context information is low, the flow executes correctly. We show that FlowCon can increase the accuracy of the context information delivered to the application up to 49%. This way, the overall robustness of the system can be increased significantly, even when faced with very low accuracy activity recognition results.

The concept of FlowCon has been extended and we applied its general idea also to the more flexible HFM. The result is *Flexible Flow Context (System) (FlexCon)* (4) [WHR11], a mechanism that leverages a combination of imperative (rigid) and declarative (flexible) flow models, Dynamic Bayesian Networks (DBN), and particle filters to reduce the uncertainty of context events. FlexCon builds probabilistic models of the dependencies between events and uses this in a similar way as FlowCon to improve the processing of probabilistic context events. Evaluation results of FlexCon show that it decreases the context event uncertainty by up to 73%. While standard flow technology exhibits a high flow failure rate of 82% when executing hybrid flows under uncertain context information, the failure rate of FlexCon is only 65% on average of all flows.

As both, FlowCon and FlexCon share the same principle we refer to both of them as ⋆Con (StarCon). We spent additional effort on both, FlowCon and FlexCon, adopting them to use more sophisticated models to represent uncertainty. The result is Hyper Flow Context (System) (HyperFlowCon) (5) [WHR13] using our novel Subjective Logic Condition Evaluator (SLCE) to combine two sources of context information in FlowPal in a systematic and effective way. One source of information are the real-world context events FlowPal receives, the second consists of the likelihood of these measured events as it is indicated by FlowCon and FlexCon. Combining both into a more meaningful piece of information that can guide the execution of flows more robustly is non-trivial. The SLCE allows us to solve this problem. With HyperFlowCon we also introduce a number of mechanisms to further decrease the impact of uncertain context information. We provide evaluation results for the HyperFlowCon and show that it outperforms a system that relies on FlowCon alone by up to 25%.

The methods mentioned so far aim to reduce the uncertainty of a single context event. In Contrast, the Fuzzy Event Assignment (FEvA) (6) [WHP11] method of FlowPal resolves errors in a sequence of context events. It enables workflows to interpret incoming uncertain context events and assign them to the correct activities in a robust fashion. FEvA exploits the structural and contextual relation of workflow activities and the current execution state of the workflow as additional information for dealing with false positive, out-of-order, and missing events. This leads to a notable improvement of the robustness. On average 91% of the context events

are correctly assigned, and the number of successfully completed flows increases on average by 52% over a reference system without FEvA.

Finally, the evaluation results of all of the proposed methods have been achieved using a two-tier evaluation approach. The first approach is based on a real-world case study that we conducted in a geriatric nursing hospital. We collected context information as well as the workflow based application in-situ and used both later to conduct offline experiments that show the effectiveness of FlowPals methods with respect to real-world data. The second approach founds on the pattern-based generation of artificial workflows (7) to drive a large scale simulation. This way, we could create a statistically significant number of structurally different flows that exhibit similar properties than the one observed in the hospital scenario. Using this set of workflows allows us to evaluate the performance of FlowPals methods independently of a specific workflow structure found in the scenario.

The contributions of the thesis author in the cited publications are the following.

In "Robustness in Context-Aware Mobile Computing" [WHR10] I first described FlowCon, analyzed the related work, set up the formal context and flow model as foundation to design, implement and evaluate FlowCon based on the highly structured blood-sample flow observed in a geriatric nursing ward.

With the next paper, "FlexCon – Robust Context Handling in Human-oriented Pervasive Flows" [WHR11], I provided a deeper insight on the flexible execution of flows in the geriatric nursing home. I extended the flow model to accommodate imperative as well as declarative modeling components within the same model. While both modeling components have been discussed earlier, their combination in a single model is an original contribution. Based on this extensions, I modified the FlowCon algorithm to deal also with less structured flows, leading to better human support when flows provide more flexibility. The result is the FlexCon algorithm which I designed, implemented and evaluated.

The contributions of "Fuzzy Event Assignment for Robust Context-Aware Workflows" [WHP11] are mostly joint work with Jonas Palauro, whom I supervised during his diploma thesis. I provided the general idea of having a flexible and adaptive mapping of events to respective flow activities with fuzzy logic, supervised the algorithm design, made some modifications to improve handling of deleted events and performed the evaluation and analysis of the published results.

Finally, in "Dealing with Uncertainty – Robust Workflow Navigation in the Health-Care Domain" [WHR13], I summarized the already published concepts under a common architecture – FlowPal– and provided analysis and results on the impact of the used model of uncertainty to represent context information. For the integration of Subjective Logic (SL) and 4V in FlowCon I was supported by Michael

Trunner, supervising his diploma thesis. The extensions of minimal event ignorance and the adaptive navigation threshold have been proposed by him. I have further integrated SL and FlexCon and performed the evaluation and analysis.

The data from the geriatric nursing home required to perform the evaluation have been collected mostly by Agnes Grünerbl and processed by Gernot Bahle, Tobias Unger and myself. I contributed to the study setup and collected some of the data first hand. For each publication Klaus Herrmann supervised and improved my work, guiding me to define the contributions in precise and cohesive way.

1.3. Structure

Overall, the thesis is structured in four parts. In Part I we present background information (Chapter 2) that is relevant in the scope of this thesis. In Chapter 3 we introduce our system model that is the formal foundation for the concepts and algorithms developed. They are described in detail in Part II of this thesis. First we introduce the system architecture of FlowPal in Chapter 4 and the Hybrid Flow Model in Chapter 5. After that we present the algorithms to reduce uncertainty, ⋆Con and its variants, in Chapter 6 and then FEvA and its algorithms for adaptive event assignment in Chapter 7.

Having described all the theoretical information on the concepts and algorithms, we evaluate their quality and performance in Part III. Chapter 8 describes our two way evaluation strategy, that is based on a real world scenario in the health-care domain and thorough simulation experiments. The actual results are presented and discussed in Chapter 9. Finally, we conclude and summarize our findings and finish with an outlook on future work in Part IV

2. Background

As the focus of this work is placed between the area of business process management and context-aware mobile applications, we want to further introduce the relevant background for each of these areas. We present a basic overview on the state of the art in each area, clarifying the most relevant terms and principles. Then, we address the relevant problems in the scope of this thesis. This way the relation of FlowPal and its mechanisms to each field of application will become clear in the following chapters. We start giving a general overview on the area of business process management (BPM). This includes the general life-cycle of a flow, the modeling languages to create them and the paradigms behind those languages. After that we focus on context information in modern mobile computing systems, with a special focus on two topics. The first one covers abstract high-level context events, which are suitable to drive the execution of an equally abstract business process. The second topic is the representation of uncertainty for this kind of context events.

As the results presented in this thesis have been developed as part of the ALLOW project, we also relate them to the project goals. ALLOW aims to promote so called adaptable pervasive flow as next generation programming paradigm for pervasive applications. Based on this vision we conclude the chapter discussing the application domains and challenges tackled in ALLOW.

2.1. Business Process Management

We have already seen that the classical application scenario for automated execution of business processes is in the "paperwork industries", such as insurance companies and financial industry. Before BPM became a standard technology in these domains, there used to be a number of software tools to support the business cases, application forms and other documents. Each of these tools had been used for individual tasks in the overall business process. They were optimized for these tasks and worked mostly independent of each other.

By introducing the workflow programming paradigm two goals were pursued. First, to bridge the gap between the tools allowing them to be orchestrated to achieve a common higher goal. Second, to automate the processing steps between

the tools and even automate some tasks to full extent. This way business processes could be executed spending less human resources. In general, the workflow concept is suited best for applications that exhibit a low flexibility but require a high throughput for individual cases.

The workflow describes how the tools should "play together" (orchestration) and what has to happen in the environment to achieve the goal of the application (e. g. for an insurance to process a customers damage report). But a workflow does not implement algorithms or user interfaces. These are (usually) already provided by the available tools. Hence programming with workflows is considered to be programming in the large. The workflow is just for structuring tasks, automating tools (services) and also to control the involvement of certain stakeholders in a company (i. e. the involvement of humans in its execution). As automated processing of workflows became state of the art in technology a large number of models and concepts have been developed.

2.1.1. Flow Life Cycle

In order to structure these concepts, we start to explain the overall life-cycle of a workflow-based application. It is generally divided in four distinct phases. These are 1. the modeling phase, 2. the deployment phase, 3. the execution phase and 4. the auditing phase. To further elaborate on each phase of the life cycle, we will have a closer look at a standard sample application, namely a credit card application workflow [YAW13].

First, we need a domain expert that acts as workflow modeler. He creates a *flow model* $\mathcal{F}$ of the workflow in the modeling phase. The flow model $\mathcal{F}$ – formally introduced in Definition 5.2.1 – is a template for a specific type of workflow. The modeler uses some workflow modeling language to create the flow model. Actual modeling languages and their properties will be discussed later in this chapter. The flow modeler starts to describe the behavior of the workflow application by breaking it down into several activities. An activity represents some computational task, that is either executed directly within the workflow, is performed by using a (Web) Service/Software-Tool or can be completed by a human (with tool support) in the real world. Then, the modeler structures the activities according to the desired application behavior.

Typical flow modeling languages provide two ways to structure the activities: control flow and data flow. The control flow defines the ordering of the activities from a business point of view. It is required to achieve the desired application semantics. Modeling the data flow explicitly defines data dependencies between existing

activities. These dependencies may be totally different from the control flow. Usually the data flow provides less structure for the workflow, because some of the activities do not share data and therefore are not ordered logically. While either type of structuring activities is valid, modern flow languages usually rely on explicit control flow only. The data flow is derived from the data dependencies, which are defined by the control flow implicitly.

For the example application the flow modeler creates the following list of activities:

- receive application
- get more information
- check application
- check loan amount
- check for small loan amounts
- check for large loan amounts
- make decision
- start approval
- notify acceptance
- deliver credit card
- complete approval
- notify rejection

Then, the flow modeler structures the activities according to the desired application behavior. Figure 2.1 shows the control flow of the credit card application workflow, only. The data flow is not modeled explicitly. In this example the activity "deliver credit card" has a data dependency – the address where to send the card – to the application form activity "receive application" but it has no data dependency to either of its predecessors, the checks for the loan amount.

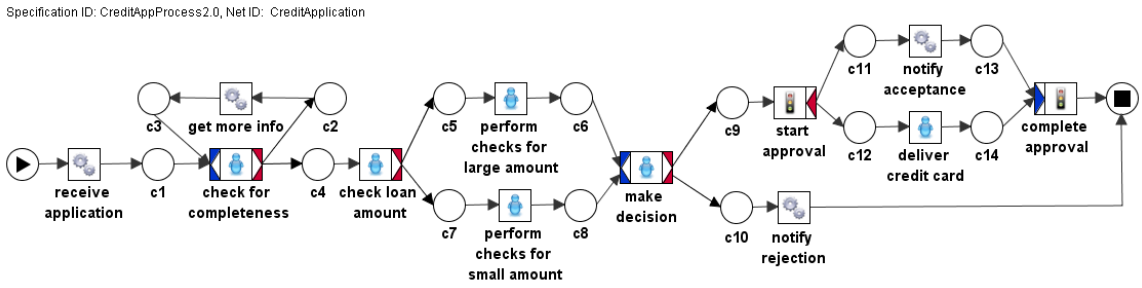

Figure 2.1.: control flow of the credit card application workflow [YAW13]

Also this example does currently not depend on context information. The workflows we focus on in this thesis are all context-aware. Therefore, the integration of context information in the flow model is very important. We will discuss this issue in more detail in Chapter 3, where we introduce the formal context model. In conclusion, besides the domain expert another expert for context management should be involved in the flow modeling phase to design the workflows. At the end of the modeling phase the result – the flow model – is a complete template for an flow-based application that can be executed. The flow models are usually stored

in repositories, where companies manage and maintain their intellectual property on business processes.

In the deployment phase the flow model is installed in a sufficient IT infrastructure, so that the flow can be executed. A *flow engine* is required to coordinate the execution and all the services defined in the activities must be bound to the flow model. In order to facilitate the reuse of processes the modeling languages usually allow to abstract the actual binding of a service from the flow model. This allows to decouple the actual services used by the workflow at run-time from its deployment environment and the available services. Before the flow can be executed, the external service requirements must be solved. This can either be achieved by binding the process statically to services at deployment time or by providing the flow the necessary functionality to discover and bind the services at run-time (late binding) [CK10]. For both methods an enterprise service bus is required to provide a basic communication and collaboration infrastructure for the workflow and the services.

Considering our sample application, we would need things such as:

1. a flow engine to execute the credit card application flow,
2. a tool, such as a web application, for the applicant to enter all the data for the application form,
3. a web services capable to make automated decisions (to perform the make decision activity) or inform the applicant on the result.

As we focus on running workflows and their interaction with context information, the deployment of a workflow is only of minor importance. However, there is one more aspect of the deployment that should be discussed here. In classical deployment scenarios there is usually only one highly available flow engine for a given application landscape. For the execution of workflows in mobile and context-aware computing scenarios different other deployment variants can be considered. To achieve a high responsiveness the workflows can be deployed directly on one users mobile device. A suitable flow engine– Sliver – has been introduced and evaluated by Hackmann et al. [HHGR06, HGR07]. Furthermore, the deployment of the workflow could be adapted at run-time in order to improve the experience of a human user interacting with it. Appropriate algorithms to achieve this has been investigated by Hiesinger et al. [HFF+11].

Whenever a workflow has to be executed in the execution phase, a runnable *flow instance* of the flow model (e. g. for a specific case of the flow) must be created. Then this flow instance can be executed by a flow engine. To create a flow instance f of a certain flow model $\mathcal{F}$ we define an instancing operator $f = \mathcal{I}(\mathcal{F})$. While executing a flow instance the flow engine is responsible for handling the data flow and

control flow and further executes the individual activities. To do that, it uses information stored in the flow instance, user input, external events and information from the services.

To coordinate the data flow consists the flow engine must convert the results from some activities to match the input format of following activities. The control flow is handled by the *flow navigator*. It uses the available information to make navigation decisions, that lead to different execution paths in the flow instance. In our example the results of the check loan amount and make decision activities have influence on the execution path. But the flow navigator is responsible to perform that decision while executing the workflow.

Activities are either executed directly in the flow engine or by invoking the respective service. The flow engine keeps track of each activities execution state. The execution of a workflow completes when the last activity completes its execution. We provide more detailed information on the activity execution and the activity execution states in Chapter 5.

The execution of an activity can also require tasks that have to be performed by human users. State of the art workflow-technology uses the concept of so-called work-lists to schedule tasks for humans. Work-lists are either individual for each person or can also be shared based on the role of the human in the organization. When a human user is involved into the execution of an activity, the flow engine creates a work-item and this item is pushed in the respective users work-list. Depending on the use case, it is also possible for the user to request tasks explicitly for his work-list (pull). A work-list always requires the human to have some device and the user must also spent some attention to interact with it. Therefore, we consider it a rather inefficient tool for the kind of applications discussed in this thesis. We elaborate on this topic further in Section 5.1 of Chapter 5.

Also, the information described here provides only a general overview on the execution semantics of a flow-based application. We discuss flow navigation and flow execution semantics in more detail in Section 5.2 of Chapter 5, where we introduce the formal flow model we use in this thesis.

In the final auditing phase of the workflow information on the execution of flow instances is archived. This auditing information is used for a number of different purposes. First, it documents the execution of the flow instance. Second, it can be used to perform posterior checks on the business case, such as compliance rules. Third, the information can also be used create business reports on the handled cases. In the scope of this thesis, the auditing information is used to inform FlowPals methods on previous executions of the same flow model, but otherwise is of little importance.

2.1.2. Flow Modeling

As the flow model later is the key source of information for FlowPal, we want to elaborate on the flow modeling concepts. We introduce the general flow modeling paradigms used today to create workflow-applications. For each type of modeling language we will also show common representatives. After that, we cover briefly workflow-patterns as means to assess the capabilities of a workflow modeling language.

A workflow modeling language is basically used to structure the partial execution order of activities. The flow modeling paradigm defines essentially how this structure can be created. First workflow modeling languages used the *imperative* workflow modeling paradigm. This paradigm is informed by imperative programming languages. It provides strict control on the execution order of each activity. Each transition between any two activities must be modeled explicitly. Only if a control flow dependency between any two activities is defined explicitly, a subsequent execution of the second activity is permitted. To some degree, imperative modeling languages allow fully unconstrained execution of activities. To do this the activities must be grouped together explicitly without any form control flow. The imperative workflow modeling languages usually cover the modeling of data and control flow, where the data flow can also be modeled implicitly.

Two common representatives for imperative workflow modeling languages are the Business Process Execution Language (BPEL) and Yet Another Workflow Language (YAWL). BPEL is the de facto industry standard to model automatically executed business processes. It is part of the Web Service Stack (WS) and commonly used for orchestration in Service Oriented Architecture (SOA). Hence, BPEL is a heavy-weight very feature rich and standardized workflow modeling language. It supports transactional behavior during workflow execution. BPEL also provides expressive fault, error and exception handling, and allows to compensate activities. The modeling semantics of BPEL can be mapped to Petri Net (PN)s as a suitable formal foundation (cf. [LVOS09]). BPEL is open for extensions so that further modeling elements may be added. To interact with external services it relies on the appropriate protocols from the WS such as the Web Service Definition Language (WSDL). YAWL [AH03] is an open source scientific alternative. It is the most expressive language to model workflows, supporting detailed modeling of control flow and data flow. YAWL has a well defined formal foundation based also on PNs, that allows an unambiguous and automated formal verification of the created workflow models. Furthermore it provides exception handling for design time as well as run-time errors. To cope with flexibility in workflow models, YAWL has been extended multiple times. One additional feature are the so-called worklets designed by Adams et al. [AHEA06]. A worklet represents a number of variant sub-processes for a task that can be exchanged dynamically at run-time.

YAWL also has been extended with a declarative workflow modeling language – DECLARE – by Pesic et al. [PSSA07]. In contrast to the imperative paradigm, it allows the modeler to provide guidance instead of explicit orders. Other examples of declarative workflow modeling can be found in Leymann et al. [LUW10] with a focus on person-centric flows and Lu et al. [LSPG06] with a focus on timely scheduling of activities. Using a declarative workflow model basically any execution order between the activities is allowed. The modeler defines constraints that prohibit the execution of invalid activity sequences. The constraints can also enforce some transitions during the execution. A model can be structured well using just a few constraints. On the downside it is hard for the flow modeler to predict undesired execution sequences at design time that would require further constraints to prevent them.

There is another paradigm that has to be mentioned in the scope of workflow modeling paradigms. Workflows are modeled naturally by business experts who are not necessarily IT-specialists as well. Profound domain knowledge is a key requirement for precise modeling results. Therefore, flow modeling must be made accessible to the available experts. This is achieved by providing tools that enable the modelers to create the workflows using a graphical notation. By providing expressive tools, the domain experts can develop flow models more easily. Visual flow modeling allows to design workflow-models based on a well defined visual representation. This representation can then be translated automatically into an executable flow model representation. The Business Process Modeling Notation (BPMN) is a state of the art graphical modeling languages for workflows. It allows non IT-specialists to create workflows using a graphical language rather than designing the process manually. Models designed in BPMN can also be converted to BPEL or YAWL.

We already motivated in the introduction that modern workflows must become more flexible, so that human real-world interaction can be coupled with workflow execution. Workflow modeling languages must provide the respective tools to facilitate this. For imperative modeling languages this is difficult to achieve because the designer must provide the required flexibility and execution alternatives at design time already. Although, the worklet approach in YAWL allows a modeler to provide certain flexibility in an expressive way.

Declarative flow modeling allows greater flexibility. But this kind of flow models are harder to control. Modeling a workflow using only constraints can allow an unexpected execution sequence easily. In order to overcome these drawbacks YAWL further facilitates to use both modeling paradigms in an interchanging fashion. Parts of the flow that require more flexibility are modeled using DECLARE, for the more rigid parts YAWL and worklets are used. In this case, it is up to the modeler to decompose the model into different application layers and choose for each layer which paradigm is suited best to structure the application and describe the

business process. This is a very challenging task, especially when decomposing the workflow yields an application that is not natural for the interacting human user. In Chapter 5, where we introduce our Hybrid Flow Model (HFM) we show that it is possible to combine both modeling paradigms directly, getting the best from both of them. The HFM allows to provide more flexible case handling for humans when compared to basic imperative modeling but also more structure and simpler modeling when compared to purely declarative flow models.

In order to assess the capabilities of flow modeling languages independently so-called *workflow patterns* are used. They allow to make the different languages more comparable. The patterns also provide some framework to do performance benchmarks of flow engines. A workflow pattern expresses functionality on different aspects of flow modeling in a grouped representation. They relate to workflows like design patterns to traditional imperative programs. In order to implement a given pattern, a workflow modeling language must be able to express the semantics of the pattern using its own modeling elements[1]. There are a number of different types of workflow patterns described in literature, where the following ones are relevant to this work.

1. control flow patterns describe common structures in process control such as sequences, branching, parallelism and synchronization patterns. [RHAM06] As the control flow contains most of the structural information of a workflow application it provides a common solution to the most common problems found in workflow modeling.

2. service interaction patterns [AMSW09] describe common inter-process interaction focusing on large scale complex interactions. Aspects covered by this patterns are service exposition, service refinement and service integration. But also the common patterns to integrate human users as proposed by some standards (e. g. WS-HumanTask [AAD+07]) are covered by those patterns.

3. data flow patterns represent the other way data is utilized in workflow languages. This covers concepts such as data visibility, definition of data blocks, data dependencies and interaction between activities [RHEA05]. But these data flow patterns are based on certain data only. Any evaluation of uncertain information has to be performed before the data is used in the flow either to complete activities or to control the execution of further tasks based on the existence of some event or data item.

In this thesis we will make use of workflow patterns as part of our evaluation strategy. The details on this usage are described in Section 8.2.1 in Chapter 8 where we describe our evaluation methodology.

[1] Please note that YAWL was designed specifically so that the common workflow patterns could be rebuild easily. Hence, YAWL claims to be the most expressive modeling language.

2.2. Context Information and Context Management

In general context information provides information of its relevant surroundings to an application. The most common and widely accepted definition of context has been expressed by Dey and Abowed [DA99].

> Context is any information that can be used to characterize the situation of an entity. An entity is a person, place, or object that is considered relevant to the interaction between a user and an application, including the user and applications themselves.

Context information can be divided in *primary* and *secondary* context [RDD+03]. Primary context information is used to identify an entity uniquely in a concrete situation. It consists of three pieces of information:

1. the location of an entity, yielding where it is,
2. the time, when that entity occupied this location,
3. the identity, who or what entity

In combination the primary context information can be used as a basic index to query the who when and where of an entity. Secondary context information then contains any derived or additional information, which is available for a given entity in a situation coarsely defined by the primary context. Examples for secondary context information are speed of an entity, its mode of locomotion and its current activity. E. g. there is a pedestrian crossing the street while using his smart-phone ($2m/s$, walking, surfing the web), and at the same time a co-located driver in a car ($20m/s$, driving, actively driving the car).

Further, we distinguish between basic and *abstract* high-level context information. For example, a microphone provides basic context information on sounds in the environment. But it is not capable to recognize speech on its own. In order to do that, a lot of additional information is required.

Basic context information is measured from any kind of sensor or is directly available on a mobile device. The basic context information from sensors can be affected by the uncertainty of the measurement system. Also, there are no additional assumptions included in the sensor data. A GPS sensor providing outdoor position information provides basic location information and velocity, but cannot determine the actual mode of locomotion by position and velocity information alone. There are some exceptions due to physical constraints, which can be derived with little additional knowledge. For example, it is unlikely that a human travels at speed greater than $10m/s$ without a car or similar technical aid. But considered strictly, this is already additional information and a combination of information has already occurred.

The data provided by sensors is always uncertain to some degree. This is represented in terms of accuracy and precision. A good and reliable measurement system must be accurate and precise. Accuracy determines the deviation to a reference value (i. e. the true position) and the precision as metric for the standard deviation of repeated measurements. That means, the lower the standard deviation of a repeated measurement the more precise the measurement system. This is also illustrated in Figure 2.2. For example, the GPS-signal achieves avg. accuracy of 2-3 meters depending on the exact method used. The precision of the system varies depending on the actual situation. In urban areas with very high buildings, the precision of a GPS-signal can be rather low.

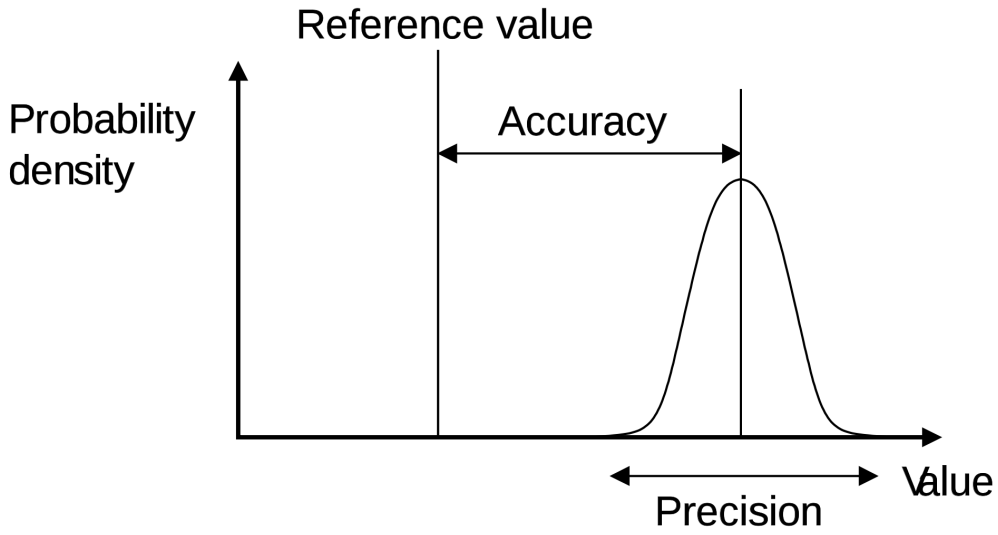

Figure 2.2.: Graphic illustration of accuracy and precision [Wik]

In addition to basic context information form sensors there is also the mobile device and its internal state as source of information available. Examples are the user account logged in, the active energy profile and the state of running applications. Overall, basic context information can only guide the adaptation and execution of an application to a limited degree. It usually has not the right level of abstraction to be useful for the more abstract context an application is executed in.

2.2.1. Abstract Context Information

In order to provide more abstract high-level context information to an application it is combined using any source of basic context information. Combining several modes of basic context information with available background knowledge allows to achieve higher-level semantics. The fusion of the GPS-position with map data can yield additional information such on the users context such as his location is 'at home' or 'at work'. Enriching the sound recorded by a microphone with information on syntax, semantics and phonetics of human language allows for speech recognition. The combination of context information becomes more difficult the more abstract the information is. Small deviations due to uncertainty in basic sensor information then can lead to different results in the combination process and to ambiguous results. Also the possible result space for combined information grows rapidly. Therefore, the availability of enough additional information to limit this growth can lead to different results.

There are numerous applications and scenarios that require abstract context information which have to consider the uncertainty in the basic context information. One application domain is the area of complex event processing. There, (context) information from different sources are combined using a set of rules to derive more abstract information. As these sources also can include uncertain sensor information, uncertainty aware solutions to complex event processing have been proposed (e. g. [KKR08]).

Another area is activity recognition, that tries to reconstruct the actual activities that humans performed in the real world, based on available sensor data enriched with additional knowledge. The health-care application domain would benefit largely from activity recognition and numerous studies (e. g. [BBA05, NAPI+03, BTJ+10, AKG+10]) support this. The focus here is also on combining (context) information from different sources with respect to reduce the uncertainty in the resulting abstract recognized activities.

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. [BTJ+10] remark specifically that the recognition process can benefit from the knowledge of domain experts. A workflow is a very detailed representation of expert application knowledge, that FlowPal uses in various ways to improve activity recognition. Barger et al. [BBA05] 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. [NAPI+03] 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 workflow provides, thus, neglecting the huge potential.

A number of general metrics to judge the quality of (abstract) context information has been proposed by Buchholz et al. [BKS03]. They describe four phases of context information processing. The phases are the sensing (1) of basic context information, the combination of context information to higher-level abstract context information in the phase of context refinement (2) , its dissemination (3) to a consuming application and its actual usage (4) . According to their work the quality of context information can be classified in five dimensions: 1. precision, 2. probability of correctness, 3. trustworthiness, 4. resolution and 5. up-to-dateness. Buchholz et al. define precision as metric how closely the measured value matches reality. As discussed earlier, in modern terms this includes the accuracy of the measurement system. The probability of correctness and trustworthiness assign a probability for its usage on the actual information and on the provider of the information. The resolution provides meta-information on the possible range of values and if it is relevant for the consumer of the context information. Finally, up-to-dateness yields a time-stamp for the context information for the consumer to judge whether it is still relevant.

In this thesis we focus specifically to improve the accuracy of the used context information and the probability of correctness. A respective *context model* must provide appropriate means to represent them both. We elaborate on this in the next section. Trustworthiness could be also factored in FlowPal but is not covered. The resolution of context is not addressed as the high-level context information for workflows is considered to be discrete (cf. Section 3.2 where we introduce the used context model). The up-to-dateness of the context information is also out of scope as only current context information are used in FlowPal. We will clarify this in the assumptions of our system model in Chapter 3. A comprehensive overview on additional aspects of context information and models can be found in the work of Baldauf et al. [BDR07], Kjaer [Kja07], Henrickson et al. [HR06] and Strang et al. [SLP04].

2.2.2. Representation of Context Information Uncertainty

The uncertainty of abstract context information can be represented using various approaches. For example the uncertainty of (continuous) position information can be described using probability density functions, or geometric representations with attached probabilities. A detailed overview on generic location uncertainty has been presented by Lange et al. [LWG+09].

We consider abstract context information as events that represent certain discrete situations. A situation is either occurring right now, or has been occurring during a well defined frame in time bound to the event. In both cases we refer to the uncertainty of this single event. The most prominent model to represent uncertainty for a single context event is Probability Theory (PT). To apply PT we must define a universe or frame of discernment Θ that contains all possible events. Then a single probability is assigned to the event $\theta \in \Theta$, with the requirement that the sum of probabilities for all possible events sums up to 1.

$$0 \leq p_\Theta(\theta) \leq 1, \qquad \sum_{\theta \in \Theta} p_\Theta(\theta) = 1 \tag{2.1}$$

The function $p_\Theta : \Theta \to [0,1]$ used to assign the probability to the events is also a probability distribution function. PT can be used to assign a degree of uncertainty to discrete as well as continuous context information. The degree of uncertainty on different context events can also be combined using the appropriate operators from PT, probabilistic and ($\vee$), probabilistic or ($\wedge$) and probabilistic not ($\neg$). Also PT has low requirements on the Context Management System (CMS) as probabilities can be made up easily from relative frequencies of the occurring events. But while this approach is relatively simple it also has some drawbacks. The combination of multiple events leads eventually to a reduced significance of the events especially when the $\vee$-operator is applied multiple times. Also PT cannot handle ignorance, i. e. the lack of information. Even when there is no evidence for any event $\theta \in \Theta$ a valid probability distribution must provide a then arbitrary value for each event. In this case usually an uniform distribution is chosen but the probability of an individual event then only depends on $|\Theta|$ and not on any evidence.

Already in the 1960/70-ies Arthur Pentland Dempster and Glenn Shafer created the Dempster-Shafer Theory of Evidence (DS) [Bar81]. As a generalization on classical PT it allows also to express ignorance on some situation θ. Therefore, not one but two values are assigned. These values represent the minimal belief in $Bel(\theta)$ and its maximum plausibility $Pl(\theta)$. For each situation these two values define an probability interval for this situation. Belief and plausibility are both defined based on a basic belief assignment.

Definition 2.2.1 (Basic Belief Assignment) *Given a frame of discernment Θ, the function $m_\Theta : \mathcal{P}(\Theta) \mapsto [0,1]$ defines a basic belief assignment (bba) under the following conditions.*

$$m_\Theta(\emptyset) = 0, \qquad \sum_{x \in \mathcal{P}(\Theta)} m_\Theta(x) = 1 \tag{2.2}$$

Here $\mathcal{P}(\Theta)$ denotes the power set of Θ and $x \subseteq \Theta$.

Having a bba the belief function is defined as

$$Bel(x) = \sum_{y \subseteq x} m_\Theta(y) \tag{2.3}$$

with $x, y \in \mathcal{P}(\Theta)$. Effectively, the belief function adds up all evidence that is contributing to the situation x. Hence, the belief is the lower limit of the probability interval.

The plausibility function is either defined as

$$Pl(x) = \sum_{y \cap x \neq \emptyset} m_\Theta(y), \qquad y \subset \Theta \tag{2.4}$$

or simply as $Pl(x) = 1 - Bel(\bar{x})$ with $\bar{x} = \Theta \setminus x$. It defines the upper bound of the interval, i. e. summing up all evidence that does not contradict the given situation. When there is no ignorance in the bba, the result of $Pl(x) - Bel(x)$ equals zero and the result is a classical probability distribution, which also shows that DS is a generalization of PT. However, there is still discussion on the interpretation and the width of a probability interval (cf. [Sha92, RN02]).

DS also provides operations to combine the evidence from multiple situations that are based on the same frame of discernment Θ. This combination rule is known as "Dempster's rule of combination" [Bar81]. But the rule has some drawbacks. The effort to combine two bbas grows exponentially with the size of Θ, so the rule quickly becomes computationally infeasible. It also can lead to counterintuitive results if there is a high degree of conflict in the situations combined (c.f. [Zad86]).

Subjective Logic (SL), a modern and even more generalized approach in representing and reasoning on uncertain information has been developed by Audun Jøsang (cf. [Jøs97, Jøs11]. It extends the classical PT with a concept to model ignorance explicitly in so called subjective opinions. Furthermore, it defines a set of combination and implication rules to reason on these subjective opinions. As we will use SL later in for FlowCon and FlexCon, we provide the details in Chapter 6.

Each of the discussed models provide us with means to express the uncertainty on a single context event. To some degree they also allow reasoning and combination of situations described by the context events. Finally, we also want to refer to a model that can be used to reason on the (possibly contradicting) situations multiple context events describe. The four-valued logic (4V) originally created by

Belnap Jr. [BJ77] is a generalization on binary logic, that also takes missing or contradicting information into account. Again, we refer to Chapter 6, where we also use 4V to reason in FlowCon and FlexCon.

2.3. The ALLOW Project

The ALLOW project propose the Adaptable Pervasive Flow (APF) [HRKD08] as suitable programming paradigm for workflows running in pervasive computing environments. An APF is basically similar to a classical workflow. But it is designed so that it is also situated in the real world and adaptable. A situated flow is logically or physically connected to some real-world entity like a human user or a mobile object. To adapt its behavior to the real-world entity the APF must be aware of its own and the entities context. As human users as well as mobile objects change their context often and unexpectedly a flow has to adapt to frequently. But due to the structure of the flow, containing information on future tasks and actions, adaptation can be proactive and hence more stable and predictable from the perspective of the user.

2.3.1. Flows and Adaptation

To guide the adaptation an APF has additional meta-data defining the *goals* and *constraints* of the flow. In order to achieve its goals, the flow can partially change its execution behavior, with respect to the defined constraints[2].

The adaptation of the flow model can either only influence the execution of the current instance (short-term adaptation) or change the flow model for all later instances (long-term adaptation). An example for short-term adaptation would be the replacement of a service in a running flow instance, with a different version or even a different sub-process. As this kind of change affects only individual activities, it is generally referred to as vertical adaption. When also a structural change of the flow is required to adapt it to the user's needs, this is considered a horizontal adaptation. If specific short-term adaptations are applied frequently, this can lead to long-term adaptation of the flow model, where the most suitable parts of the flow are used to create an optimized model in an evolutionary fashion.

[2]Please note that these constraints are different from the constraints described for declarative flow modeling.

2.3.2. Adaptive Flow Control

The flow instance is frequently interacting with both, mobile human users and services at the same time. While being executed by a flow engine, it must also adapt its own execution to changes in the environment.

Therefore, adaptive flow control pursues two things. First, the execution of the flow in a dynamic environment must be maintained. Usually, mobile users and other real-world entities change their state quickly and unexpectedly for the system. Also a failing service or a deteriorated network link may require changes to the execution and additional vertical adaptation. Second, the interaction with the user must be as smooth as possible and basically all adaptations must be transparent from the human users perspective. In order to gain the flexibility to cope with these changes, a flow instance cannot be executed in a central system. The execution of single activities must be distributed to multiple devices. This distribution can then be adapted according to utility metrics, such as energy consumption [FFHR11] on the mobile devices or the latency perceived by the human user [HFF+11].

Another aspect that is relevant in this context, is the actual task list of a specific human user. A human user tends to organize his work differently than defined in a process description. Usually, the user groups a number of similar but small tasks together to execute them in a batch. For other more complex tasks they may prefer to split them. This kind of operation can be used to adapt task lists and modify the sequence of tasks – the *person-centric flow* – of a human user. This as well as task list management in pervasive environments have also been considered in the scope of ALLOW [UELW10].

Performing no explicit interaction with the user is another means to reduce the obtrusiveness of an APF. Therefore, the activities of the user have to be sensed using activity recognition and processed by a CMS. Sensing the users activities accurately to control the execution of the flow and adapt it accordingly when required. FlowPal is providing the tools to do this.

2.3.3. Scenarios

The ALLOW project researched two different application scenarios in detail. The first scenario covers a logistics workflow in an oversea port responsible for car shipments. The scenario involves unloading, storing, maintenance task such as repairing or cleaning, and customization of the cars. The cars arrive in large quantities in a just-in-time logistics flow. This scenario provides a lot of challenges for

flow adaptation as there may be sudden changes in the process (e. g. due to a damaged car so that additional maintenance is required) or a missing services (e. g. a train for further transportation is late).

The second scenario is situated in a geriatric nursing home, where caretakers support patients, which require full-time care. The scenario is suitable for a flow-based approach as it is repeated on a daily basis per patient, in a structured fashion. In this scenario humans are in focus of the flow execution and especially the interaction between humans drives the execution of the flow. This makes activity recognition of the human tasks the most important information to keep the flow informed on its entities context. The scenario is cost sensitive so that only few and cheap sensors can be applied to recognize activities. The possible gain using a flow is significant. It could accomplish the documentation of the process and check the adherence of care guidelines, while saving some time of the caretakers, usually spent for manual documentation. This is also the scenario, we evaluated FlowPal against. Hence, a more detailed description, our findings and the evaluation methodology are all elaborated in Chapter 8.

3. System Model

Before we start to describe the details of FlowPal, we have to clarify the system model it is based on. In the next section we cover all the assumptions we make, including the general types of application, the user behavior, the available devices and their means of communication as well as the means of sensing and user mobility. This way, we clarify the focus of the work presented in this thesis further, and give the foundations to introduce the system architecture in Chapter 4.

The abstraction, modeling and availability of context information have been discussed already. As the usage of context information is also a part of the system model we introduce the formal context model used in the remainder of this thesis in Section 3.2. The model aims to achieve two goals. First, it tailored for the representation of high-level abstract context information that drive the execution of a workflow. Second, it provides a flexible abstraction of context information uncertainty that we later use to investigate the usage of different uncertainty models, besides probability theory.

3.1. Basic Assumptions

First of all we consider applications that support real-world processes or activities. More specifically, if an application containing a mixture of real-world activities and computational activities can be modeled using the the workflow programming paradigm, we support it. The general goal of those applications should be 1. to structure the activities in the real world and 2. support the users to adhere to the defined process. The usual execution environment we consider, is a workplace such as the ones in health-care and logistic domain described in the scenarios in Section 2.3.3. There, one or more human users are supported by the flow-based application in their daily workplace routine. The users can be considered professionals at their respective workplace, e. g. doctors or caretakers in the health-care domain. Applications used in a private context such as in smart homes or for shopping malls are also possible but not elaborated on.

3.1.1. Process Management Execution Environment

The basic assumptions also define the primary services required by the system model. In the previous chapter we have seen that workflows are subject to a complex life-cycle and have high requirements on their execution environment. In order to execute a flow-based application, the basic components of a workflow managment system must be available. There must be some repository where executable flow models are stored. Furthermore, there must be flow engine available which can retrieve and execute a workflow. We assume that there is exactly one flow engine responsible for an application. If the actual execution of a flow instance is distributed on multiple flow engines the distribution must be handled transparent from the application point of view. When a flow requires an additional service for its execution, the service must be available.

3.1.2. User Behavior

The overall scope of flow-based applications is reduced by the kind of user behavior and interaction. For each application we assume that a significant amount of human interaction with the flow is required, to complete its execution successfully. This interaction can be either direct, using a device for example to enter data and fill forms. The interaction can also be indirect, where the flow is guided by context information of the user. We assume that the majority of interaction is indirect as FlowPal aims to allow a more unobtrusive user experience and hence indirect interaction. We will elaborate more on the topic of human-workflow interaction in Section 5.1.1 of Chapter 5.

3.1.3. Device and Network Model

In order to execute the applications in the defined environment, we require computing resources and devices. Based on the actual user behavior and role, each user may have a personal device to interact with. A device can either be mobile or static. In general each device provides computing resources. We assume that these resources are sufficient on personal devices to host the required clients for the application (system). Further, the infrastructure provides sufficient resources to host all services required for the running applications. The devices are all connected to a local infrastructure, such as a single LAN or a corporate network. The mobile devices are connected wireless. This local infrastructure spans a geographic region such as a single building or a company site, but at least the scope relevant for the application. We do not consider limited network resources or network and communication problems such as node failures, malicious behavior or network

separation for FlowPal. They are out of the scope of this thesis, because the focus lies on the robust interpretation of uncertain context information. Further, we assume that the current user of a device is always known by means such as a personal account.

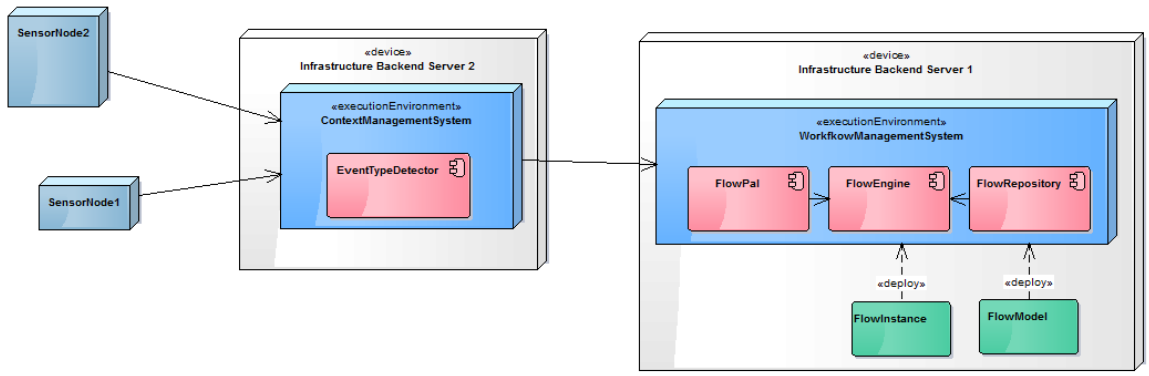

Figure 3.1.: Network Model and Deployment given only infrastructure devices

We describe two corner cases that demonstrate viable network setups. In the first case depicted in Figure 3.1.3, there are only infrastructure devices. Here the user interacts either with a stationary device also connected to the network (not shown) or indirectly, through his context recognized by the sensor nodes and processed by the CMS. The CMS can also be federated or distributed using an appropriate middle-ware. The workflow managment system here is hosted on a single server in the infrastructure, including the flow engine and flow repository. FlowPal is deployed as plugin to the flow engine. In the second case, depicted in Figure 3.1.3 there is a single mobile personal device with sufficient resources to run the application and all relevant services i. e. processing sensor data, providing them via a locally deployed CMS, that the locally deployed workflow managment system can access. Depending on the application and the actual number of humans involved in the execution, the actual setup will be somewhere between the two corner cases containing multiple mobile devices and infrastructure devices. The core of these assumptions are the following. First FlowPal stays with the actual flow engine (and may be replicated). Second, the information required to support the execution of a flow instance with FlowPal originates from the flow repository, where also the flow model is stored.

3.1.4. Sensors and Mobility

As FlowPal targets at an unobtrusive application experience, we assume that the users want to interact with the application mostly in an indirect fashion. Therefore, we assume the availability of sensors in the environment of the application

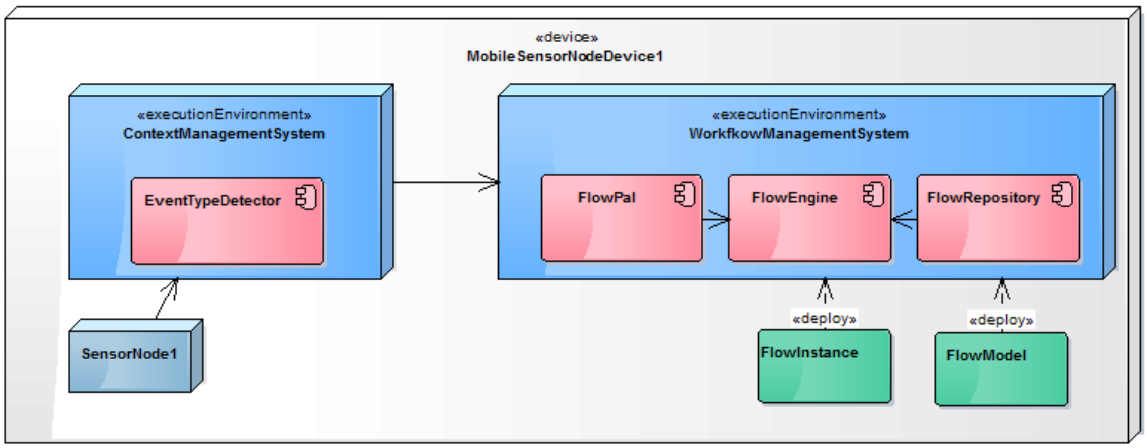

Figure 3.2.: Network Model and Deployment given only a single mobile device with sensing capabilities

that provide the necessary input to recognize the actions of the user. This can cover RFID-Readers at doors, mobile devices such as smart-phones with multi-modal sensing, or cameras in certain locations. The actual placement of the sensors and their specific type is not considered. The sensors considered to be connected to the local network infrastructure and the current readings can be obtained in a timely fashion for processing by the CMS.

The users are mobile in the geographic area defined by the application and pursue the application goal. We assume that the sensor coverage in the area is sufficient for some form of activity recognition. This way context information for the application can be measured. But this context information may be wrong, little accurate, or out-of-date. A personal device carried by a user all the time, or a an otherwise sufficient coverage of workplace are assumed as typical scenarios. A mixture of different setups may be present. This also requires that there is a CMS available in the local infrastructure, which processes the sensor data and provides high-level abstract context information to the application. As context information are the primary input for FlowPal and the respective context model, which we use in the remainder of this thesis, is of fundamental relevance, we define it more formally.

3.2. Context Model

As flows should not obstruct users in their daily routine, they are mostly or solely driven by *context events*. Context events are provided by the CMS. We assume that it exists as part of our flow execution environment. The CMS measures and

detects events in the real world (e.g. the caretaker entering a specific room) and provides these context events to FlowPal. 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 (c.f. Section 2.2.1). While the former technology might be applicable in high-cost environments such as an operating room in the health-care domain, we have to rely on the latter kind in most real-world situations.

3.2.1. Context Information and Events

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

Definition 3.2.1 (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, *person walking* could be an event type. Events of this type are created whenever a person changes her mode of locomotion to *walking*. Event types that represent semantically similar context and have the same level of abstraction can belong to a common *event type set*, and each event type belongs to at least one event type set.

Definition 3.2.2 (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 persons locomotion modes could be, e. g. $E_\alpha = \{$*person walking*, *person sitting*, *person standing*$\}$. The purpose of an event type set is twofold: First, it allows an application designer to select the most appropriate context a certain part of the application should respond to in a simple fashion. In terms of flow modeling, a single activity in a flow can react to one or more event type sets as the flow modeler seems necessary. Also the granularity and composition of event type sets can be adjusted to a specific application. We present the formal usage of event type sets in flow models and the integration of context information later when we introduce the flow model in Chapter 5.

The second purpose of an event type set founds on the grouping of the event types. Due to the related semantics of all event types in an event type set they allow 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 an flow-based or other application is executed, it can register for different event type sets of interest.

3.2.2. Event Instances and the Uncertainty Model Abstraction

Whenever the CMS recognizes an appropriate situation in the real-world it notifies the register application by sending an *event instance*.

Definition 3.2.3 (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 possible 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 $\sum_{u \in E} I_E^e(u) = 1$.*

This also means, for an event type set E, the event instance e can represent any event type $u \in E$. As the recognition relies on (uncertain) sensor readings, processing and composition of low-level context information, an event e is always uncertain. 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 $u \in E$. As we use this definition for our CMS we assume that it can provide a probability distribution for a given event type set E. For example if $u = personwalking$ and $u \in E$ then $I_E^e(personwalking) = 0.52$ indicates that the probability of e being of type *person walking* is 52%. Please note that this is only one possible way of representing uncertainty.

The definition of the event instance also yields the basic model to represent uncertainty in the context model. As the events originate from uncertain sensor data and are processed based on assumptions in the CMS they cannot be distinguished with absolute certainty. Therefore, the uncertainty must be expressed using an appropriate uncertainty model. While an event instance e on its own indicates only that some context event occurred, the actual interpretation of that context event from the CMS is defined by the abstract mapping I_E^e. In the given example, we have used probability theory, where I_E^e is a probability distribution for the event type set E and $I_E^e(u)$ is a probability. This interpretation can be changed to different uncertainty models and also depends on the capabilities of the CMS. Hence, the event e can be also seen as sample from the frame of discernment denoted by the respective event type set E. The interpretation I_E^e is then used as abstract metric to map the event to some representation of uncertainty, but not necessarily a

probability distribution. But, whatever model we choose to represent uncertainty, there is a risk that result of the interpretation I_E^e contradicts the actual real-world. One job of FlowPal is to minimize this risk by augmenting the interpretation with information from the flow.

At first, we assume for the interpretation a probability distribution based on probability theory. This allows us to reason – using probabilistic logic – on the event types of E for the given event instance e. However, this interpretation can be exchanged in a flexible fashion allowing us to study different uncertainty models later (cf. Section 6.6). For now, we will stick to PT for two reasons.

First, the interpretation of other uncertainty models (i. e. DS, SL) is not trivial and can cause unwanted side effects, such as counter-intuitive results when combining uncertain events. PT is simpler, but it leaves little space for interpretation, thus simplifying decision making. Secondly, the other uncertainty models provide additional information. But they have also stricter requirements regarding the context system, with respect to the information the CMS must provide. We will discuss this in more detail in Section 6.6, where we introduce study the use of other uncertainty models for the use in FlowPal.

Finally as the flow is executed based on a sequence of event we define it as follows.

Definition 3.2.4 (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 $\mathcal{S} = (e_1, \ldots, e_k)$ is an ordered list of k event instances, where each $e_i \in \mathcal{E}$ and $1 \leq i \leq k$.*

An event sequence $\mathcal{S}$ represents a list of context events and the temporal order in which they are received by the flow engine. This sequence is used later as formal input to a flow instance running on a flow engine. Based on the parameters of the given flow engine and the accuracy of the events in the sequence $\mathcal{S}$, the successful execution of the flow can be determined. We call $\mathcal{S}$ a *valid event sequence* if it leads to a successful execution of a given flow. The specific conditions that make an event sequence valid with respect to a flow model are given later in Chapter 5.

3.3. Discussion

In this chapter we have provided all the details to understand the system and the algorithms we introduce in Part II of this thesis. The network and device model allows for a broad range of setups for pervasive applications in professional workplaces. The used application model founds on the well known workflow principle

and has been extended to suit the needs for pervasive applications. We have defined the basic structure of workflows and their life-cycle so that we can describe exactly the places where FlowPals algorithms fit in existing systems. Finally, the context model gives us a formal frame to argue on context information in general. We have defined the event instances as the finest granularity of a context information piece. And it gives us the abstraction on the notion of context information uncertainty. With this foundation we can now describe the concepts and algorithms we developed.

Part II.

Concepts and Algorithms

4. Concept Overview

In the previous chapter we have identified and formalized the representation, interpretation and sequence issues for uncertain context information. In order to reduce the impact of these issues on the successful execution of flow-based pervasive applications we built our system, *FlowPal*, that implements the concepts we developed in the scope of this thesis.

FlowPal is structured like a toolbox. Each of its components tackles one of the mentioned issues of uncertain context information. They can be used individually or all together. Basically, FlowPal observes the incoming context events and influences their processing in the flow engine in order to reduce the effects of uncertainty. The basic principles of the components are described later in this chapter.

Each FlowPal component uses some form of *flow knowledge*. It is all kind of information that can be extracted or learned from a flow-based application. Using flow knowledge to reduce the presented issues of uncertain context information is the main idea of this thesis. We use three different ways to get flow knowledge. First, a flow model encodes information on the application semantics and temporal behavior in its structure (flow model). Second, historic data that has been collected from previously executed flow instances (flow traces and history). Third, a running flow instance provides some execution state (activity states).

The actual mechanisms to extract flow knowledge and instrument it for our purpose, depend on the tackled issue. As they are different for each component we discuss them in greater detail in later chapters, along with the concepts.

4.1. FlowPal System Architecture

In the following we present the architectural overview on FlowPal. We start with the generic flow engine architecture we presented in Chapter 2 and put it into the context of our system model (cf. Section 3.1.1). After that, we describe the basic functionality of FlowPals components and their dependencies to the existing components.

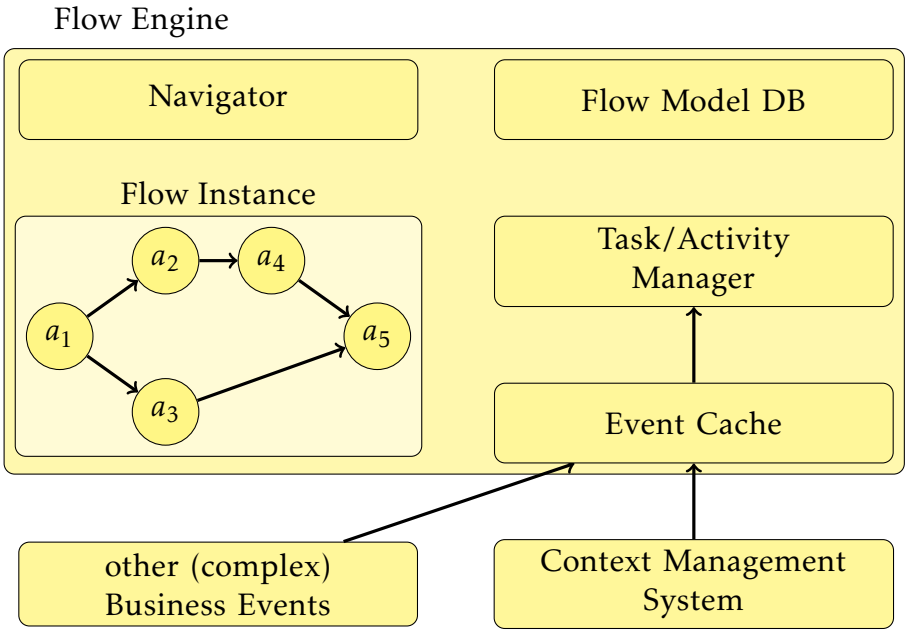

Figure 4.1.: Generic Flow Engine Architecture

The flow engine and its components, as shown in Figure 4.1, are derived from the generic flow engine architecture we introduced in Section 2.1.1, where we also introduced their functionality and the life-cycle of a flow-based application.

The first new components are the external sources of information which may either be complex business events or context events. The main difference between the two is, that the former do not suffer from uncertainty but are produced by a technical system, or represent manual human input at workstation or using a mobile device. Hence, these events are always exact. They may be wrong due to technical failures or human error, but this is out of the scope of this thesis.

The context events are all received from the Context Management System (CMS). We assume that context events are always to some degree uncertain (cf. Sections 3.2 and 2.2.1). Therefore, the CMS always provides some means to assess this degree of uncertainty.

The amount of uncertainty the application has to deal with becomes higher with the level of abstraction. While plain sensor data can be combined to reduce issues of the measured modality using assumptions on physics and the measurement system, this becomes impossible when activities must be distinguished on their high level semantics which are not known to the CMS. On the one hand, there can be a simple temperature reading from temperature sensor with a well known failure model, that can describe accurately the quality of the sensor reading and may even provide information whether the sensor has failed. On the other hand, there can be a complex real world event like a mechanic performing a maintenance task at a

machine, a nurse giving medication to a patient, or an electrician assembling some home automation. This kind of events become the more relevant the more humans are integrated into the execution of a flow-based application. As we argued in Section 2.2 it is difficult, if not impossible, to provide an accurate failure model for this kind of events. Therefore, we have to deal with a high amount of uncertainty directly at the application.

To extract flow knowledge from a flow-based application we need access to the flow model data base, where the flow engine stores the flow models. As flow models change rarely compared to the number of flow instances created from that model, it would be sufficient to retrieve the model once until it is changed and extract the flow knowledge. Retrieving the information from the flow engines auditing component to access the flow history and traces is basically similar. When a flow instance is completed and its auditing information becomes available, FlowPal should be notified about this. But, as this historic information becomes only gradually available, the flow traces should be processed incrementally by FlowPal. Finally, when FlowPal accesses the execution state of a running flow instance, it must be obtained directly from the flow engine which it is executed on.

Because FlowPal requires this deep access into the flow engine and its influence on the processing of context events also happens directly within the flow engine, the components of FlowPal are designed as flow engine plugins. The relationship of the individual plugins with respect to the generic flow engine architecture is shown in Figure 4.2.

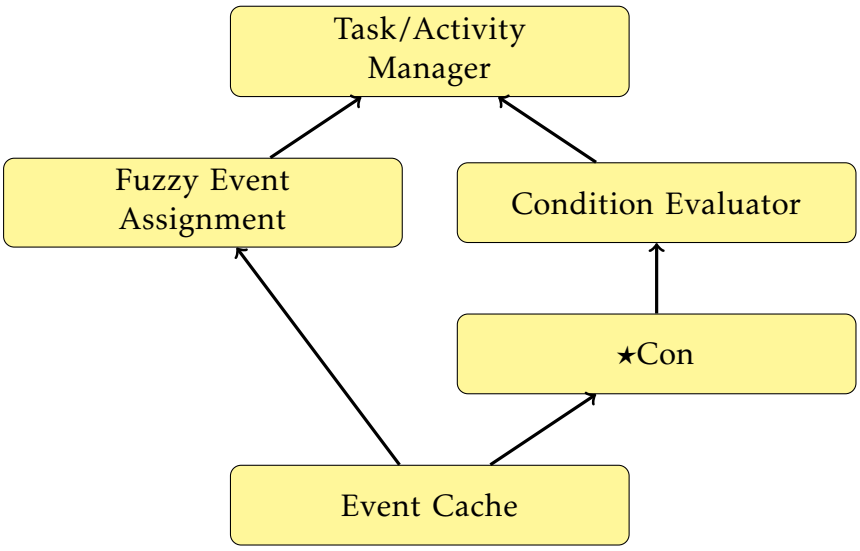

Figure 4.2.: FlowPal Architecture Plugins

Overall there are three components that may be added or altered by FlowPal. First there is the Condition Evaluator (CE). Its main objective is to provide the necessary means to evaluate the context conditions, which in turn instrument the flow navigator to make its decisions. The CE provides the operators to reason on different event instances and to perform also the required interpretation of the event instances. A basic CE which provides this functionality is required by every flow engine for reasoning on context events. We extend this basic CE in two ways. First, when using ⋆Con, the CE must also provide operations to fuse the information from an event instance with the information provided by ⋆Con on that event. Second, we employ various sophisticated models to represent uncertainty which leads to different fusion operators and also changes the logic operators to reason on the context events.

The second component is ⋆Con (StarCon) that aims to resolve interpretation issues for received context events. ⋆Con receives a copy of the incoming event instance and, based on the actual state of execution of the running flow instance, provides a second source of information for the actual context information. Basically, it judges whether the received event instance fits statistically to the current combination of execution state, user and context. We provide the full concept and all the required details later in Chapter 6. The information from ⋆Con can then be used by the CE and combined with the event instance. The means for this combination are provided by the CE.

The third plugin is the Fuzzy Event Assignment (FEvA) that provides resilience against false positive, missing or out-of-order context events. FEvA enables the flow engine to interpret incoming context events robustly and assign them to the correct activities. To achieve this FEvA, extends the state space for the activities and provides an additional cache for the context events called the *Event Container*. Using a two step algorithm, that relies on flow knowledge and the current execution state of flows as additional information, allow FEvA to deal with errors in the event sequence. This leads to a notable improvement of the robustness of context-aware applications.

In the next chapter we will introduce the formal generic flow modeling language that we used to build FlowPals components. Following that, we introduce ⋆Con and its variants in Chapter 6. There we will present also our approach optimizing the representation of uncertainty of context events. In Chapter 7 of Part II we describe FEvA in detail, and how we tackle sequence errors.

5. The Hybrid Flow Model

5.1. Preliminaries

Workflows are an adequate means for modeling the functionality and the temporal flow of complex activities. While modeling a workflow the modeling language provides abstraction from existing IT-infrastructure. This way a flow modeler can be rather a domain expert than an IT-specialist. The modeler works in his area of expertise and can focus on the business logic instead of all the technical details. This also allows to model the processes using the right layer of functional abstraction e.g. for the business model or service interaction. To further ease the modeling process a wide variety of workflow modeling languages have been created and for a lot of them there are graphical editors to create workflows with little effort.

But as the areas of application for workflow technology have increased, so also have the requirements to the modeling languages. For example, today's workflows have to support transactional processing [Gre02]. Furthermore workflows can be executed flexibly crossing the borders of different companies seamless [LGB05] and can be created on the fly using techniques for automatic service composition (e. g. [DS05]).

5.1.1. Human interaction in workflows

Workflows also have been proposed as useful tool for environments with intensive human interaction. However, workflows must deal with the great flexibility in human behavior. In general we can distinguish between two kinds of interaction between a human user and a workflow.

First, the human interacts with the workflow directly using some electronic device. For this kind of flow control, the workflow must be capable to tolerate the deviations in human behavior. Case handling in workflows as described by van der Aalst et al. [AWG05], is one way to improve support for human users. Based on the circumstances a human can decide which predefined variant of a flow is executed. However, the variants have to be modeled in advance and a decision

cannot be changed later if the situation has changed unexpectedly requiring another case to be executed instead. More flexibility is required to integrate human tasks in workflows.

One way is to integrate the human task directly in the workflow as kind of "human-service". WebService-HumanTasks is one well known specification for this kind of services [AAD+07]. It provides features to describe tasks for people, notify them and bind people to certain tasks based on their organizational role. These features are complemented by WS-BPEL4People [IKM+10]. BPEL4People supports humans to perform tasks in business processes based on their role in an organization and manages aspects like delegation of tasks, the four-eye principle, and escalation of overdue tasks. The means of interaction are not defined directly, and the human-service implemented is atomic for each task. Hence, a human can perform one or more atomic tasks in a workflow, depending on the roles and process structure. But there is no way to structure the execution of those tasks, as their order and assignment is controlled by the executed workflow, alone.

Furthermore the workflow requires the human to be ready to interact with a human task anytime and anywhere. The *PerCollab* system developed by Chakraborty et al. [CL04] solves this by determining the best device to notify and interact with the human user. They use static and dynamic information on the user's context such as instant messaging online status, calender entries, position data, etc. But again, while PerCollab facilitates the interaction, the workflow is in control.

Vice versa, the human is in full control of the workflow. There are some works for this kind of workflow interaction, which can be summarized as personal workflows. An early work on the idea of personal workflow and the interaction with them, was presented by Hwang and Chen [HC03]. They provide a model to specify queries to a personal process, that give control on the process and provide feedback to the user to execute it further. On the downside, the modeling of the personal workflow and the monitoring of its execution state rely mostly on direct user input.

The *sentient processes* presented by Urbanski et al. [UBR06] are based on a proprietary workflow modeling language for personal workflows, which are suited for pervasive computing. Such a process is tailored to a single user that wants to execute a sequence of tasks in a pervasive environment. The later extension, *PerFlows* [UHW+09], allows the user to decide in which order he executes his tasks flexibly. They can be executed automatically, rolled back, skipped and are context aware to the behavior of the owner. However, the sentient processes as well as the PerFlows allow only the assignment of a single user to the flow. The user is static and the flow can react only on context information of that user or direct input.

The personal workflows allow the human to control the workflow, but he has still to interact actively with it. As a consequence the user requires a lot attention to control the workflow.

The alternative way for a human is to control a workflow indirectly. To do that, the human interacts naturally with his environment and the workflow is executed according to this interaction. To adapt to this kind of interaction workflow execution has become even more flexible and supporting. This flexibility can be achieved using a declarative flow modeling approach [PSSA07], which we have already defined in Section 2.1. Currently, there is no flow model that allows seamless integration of both modeling approaches in the same language. The HFM enables the flow modeling expert to use both approaches simultaneously. The flexibility gained is one of the critical factors for humans to interact seamlessly with workflows. The other one is a workflow being context-aware.

5.1.2. Context in workflows

A workflow is context-aware if it has access to context information and can reacts to it in a meaningful way. However the level of abstraction of the context information that a workflow can process can vary greatly. The most basic level is again represented by raw sensor data, while on a high-level of abstraction a context event indicates that a human has performed a certain action in the real world relevant to the flow. As workflows are modeled on a high-level of abstraction the latter type of context information is much more useful for them. The context information can be accessed either directly or using a separate CMS.

The PerCollab system, discussed earlier, already uses context information to determine the best way to interact with a human user, but the human cannot control the execution of the workflow by his actions alone. An early approach based on BPEL and web services was proposed by Han et al. [HCC05]. Their ubiquitous workflow description language (uWDL) adds triples consisting of subject, verb and object to the transitions in a workflow. When the context is matched the respective transition is activated. The context information can originate from various sources and are organized in a common ontology. The authors also provided a context-aware extension to integrate services in this kind of flows with uFlow [HCKC06]. The direct integration of context information in workflows was first introduced by Wieland et al. [WKNL07]. The authors provide a set of operators to access context provided by a CMS. These operators include so-called "context events", "context queries" and "context decisions". While the "context event" and "context query" operators allow passive notification and active querying of context information, the "context decision" changes the path of a Workflow execution based on the current context. In follow up work the authors investigated further the right level of abstraction that context information must provide for workflows [WKN08]. They

also proposed an architecture that allows to integrate uncertain context information in workflow but no concrete algorithm or method that aims at improving the uncertainty of thecontext information [WKL+09]
The PerFlows [UHW+09], mentioned earlier, are context-aware workflows and provide flexible activity scheduling and processing. But they still rely on direct user interaction to work properly. This is disadvantageous in scenarios where the workflows shall run in the background in order not to obstruct the user. Further we proposed our own method, to make situated APFs context-aware and ready for adaptation based on this context information [EFH+09, WHR09].

As the uncertainty of context information is of major interest later in this thesis, please note that none of the above mentioned approaches using context information in workflows considers the uncertainty of context information in any way.

Having both, a very flexible flow model and a workflow that is context aware, provides a solid foundation to support a human in his activities without obstructing his actions where not strictly required by the application. The HFM that we introduce in this chapter is a step towards this goal. It allows a seamless usage of both - the imperative and the declarative - modeling paradigms within the same workflow model. Thus it provides even better balancing than described by van der Aalst et al. [APS09]. The HFM can be modeled using exactly the right layer of abstraction for the user. Using the HFM a flow modeler can focus not only on the business process but also strongly on the human users involved in the flow. The modeler can provide the greatest amount of freedom and flexibility to the human stakeholder where it is allowed and the necessary guidance where it is required, e. g. for an inexperienced user. This leads to flow models that allow a human flexibility in their execution, where groups of declarative activities are used and a seamless transition to parts of the flow, where procedures have to be performed to the letter, using the restrictive imperative modeling of structured activity sequences. These kind of processes can be tailored directly for the execution by one or more human stakeholders. Therefore, they are very well suited to model person-centric flows.

The HFM has been developed in different steps. We started with a basic imperative variant to show that the mechanisms of FlowPal– especially FlowCon and FEvA– are feasible for imperative workflow languages. Then we extended it to the final HFM, allowing more flexibility and also adapting the mechanisms. In order to keep the HFM simple and focused, we neglected a number of standard features e. g. exception handling and transactional behavior. But as we keep the HFM simple, we can easily show that its modeling capabilities are equivalent to the most basic and most formal flow modeling language based on PNs. Furthermore, the HFM also builds the formal foundation for the mechanisms of FlowPal we have developed in this thesis.

In the remainder of this chapter we introduce the HFM gradually. First, we describe the basic imperative variant of the HFM and its execution semantics. This provides the foundations to understand the working principles of FlowCon and FEvA later on. After that we extend the model to the final HFM, defining the additional modeling elements. Based on those we explain the execution semantics of a flow. Then, we can also clarify the relation between event sequence (cf. Section 3.2), error model (cf. Section 7.2.3) and the successful execution of a hybrid flow instance. We conclude the chapter with a discussion of the results.

5.2. The Basic Imperative Flow Model

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 their order of execution. This basic model can be mapped to a Petri Net (PN) and vice versa just like other workflow models. A sketch of this mapping is given at the end of the section. Subsequently, we define each of its components and the basic execution principle of the created flow instances. As a running example we use the small flow snippet depicted in Figure 5.1. A full example flow is shown later in Section 6.4.

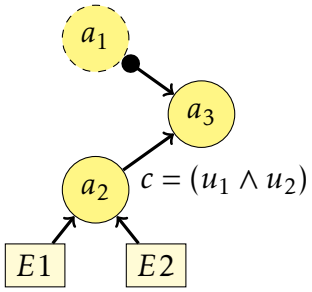

Figure 5.1.: Flow snippet showing all modeling elements of a basic imperative flow

Definition 5.2.1 (Flow Model) *A flow model $\mathcal{F}$ is a 5-tuple $\mathcal{F} := (A, T, C, \Phi, M)$, consisting of a set of activities A, a set of transitions T, and a set of conditions C, where Φ is the set of mandatory activities and M is the set of marked activities.*

$\mathcal{F}$ defines the 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 navigates the transition from

a_x to a_y. Some activities are mandatory, and must be completed for a successful flow execution. The flow model $\mathcal{F}$ acts as a template for an application. It can be used to create a flow instance that can then be executed.

Definition 5.2.2 (Activity) *An activity a represents an atomic piece of work within a flow. The activity is atomic in a sense that it must either be completed correctly or not at all for the flow to execute successfully. Activities include tasks such as 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. For the example this are $A = \{a_1, a_2, a_3\}$. An arbitrary number of event types can be added to each activity. Let

$$\epsilon_a := \mathbb{N} \mapsto \mathcal{P}(U) \tag{5.1}$$

be the event type set 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 type sets associated with a, then $\epsilon_a(i)$ yields the i-th event type set 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 the activity a. The example shows two event type sets E_1 and E_2 being mapped to the activity a_2, hence $\epsilon_{a_2}(1) = E_1$ and $\epsilon_{a_2}(2) = E_2$.

Furthermore activities may be marked as mandatory. For a given flow $\mathcal{F}$, $\Phi \subseteq A$ is the set of mandatory activities. Further, the mandatory activity function $\phi := A \mapsto [true, false]$ is defined as

$$\phi(a) = \begin{cases} true & \Leftrightarrow a \in \Phi \\ false & , otherwise \end{cases} \tag{5.2}$$

The example has two mandatory activities $\Phi = \{a_2, a_3\}$, which can be distinguished from the non-mandatory activity a_1 that is drawn with a dashed circle.

Finally, when an activity is executed it assumes a number of distinct states. These are in their order of execution $Z = \{inactive, ready, active, complete\}$, where Z denotes the overall *activity state space*. Further, let ω be the function that retrieves the current state of an activity a.

$$\omega : A \to Z \tag{5.3}$$

More complex models for activity states exist e. g. for compensation or exception handling but for the focus of this thesis the simple approach is sufficient. In Section 7.3 we will discuss the influence of uncertain context information on the activity states in greater detail.

The activities are structured and ordered by transitions.

Definition 5.2.3 (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

$$d_{in}(a_i) := |\{(a_x, a_y) \in T | a_i = a_y\}| \tag{5.4}$$

and

$$d_{out}(a_i) := |\{(a_x, a_y) \in T | a_i = a_x\}| \tag{5.5}$$

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. In the example there are two transitions (a_1, a_3) and (a_2, a_3).

Conditional decisions on transitions can be made taking the context conditions into account.

Definition 5.2.4 (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.*

A transition $t = (a_x, a_y)$ can be annotated with exactly one condition c_t. This condition c_t is evaluated when a_x has received an event instance e for every ϵ_{a_x}. We apply the received event instances to c_t and check

$$c_t[u/I_E^e(u)] \geq t_n \tag{5.6}$$

against the *navigation threshold* $0.0 \leq t_n \leq 1.0$ of the flow engine. In the example the condition at the transition (a_2, a_3) is $c = u_1 \wedge u_2$. The condition is evaluated when a_2 has received an event instance for $E1$ and $E2$, where the event type $u_1 \in E1$ is part of the event type set $E1$ and also $u_2 \in E2$. The actual value the condition uses for evaluation is given by the interpretation of the event instance as denoted in Equation 5.6. After the substitution the condition can be tested against the navigation threshold.

At this point the flow engine makes the actual navigation decision and removes further uncertainty for that decision. Please note, that the flow engine defers this decision to the latest time possible before actually navigating the flow.

If the equation is fulfilled, the condition evaluates to true, the transition is activated and the activity a_3 is executed. According to definition 5.2.4 and definition 3.2.3 (event instance) we use PT to evaluate a condition. However, in Section 6.6.1 we will elaborate on the used uncertainty model and present alternative approaches to the basic one defined here.

Finally we introduce the transition marker as means to control the join 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.

Definition 5.2.5 (Transition marker) *Let $M \subseteq T$ be the set of all transitions in a flow $\mathcal{F}$ that have a transition marker. The transition marker function $\mu := T \mapsto [true, false]$ 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.*

In flow diagrams, the markers are denoted as additional dots at the origin of transitions. The example shows a transition marker at (a_1, a_3). So in order to execute a_3 it is sufficient that only a_2 has been executed and the condition c has been evaluated to true. If the mark would be missing, also a_1 is required to complete its execution before $a3$ can be started. This would also make a_1 (transitively) mandatory.

5.2.1. Basic Execution Principle

Having all the basic components of the flow model defined, we can continue to describe the way it is executed.

First, we need to create a flow instance f of the flow model. Second, there must be some event sequence S that can drive the execution of the flow instance. Finally we have to define the flow engine that actually executes f.

We can create a flow instance f from $\mathcal{F}$ using $f = \mathcal{I}(\mathcal{F})$ and execute f using a basic flow engine, that is defined in the following.

Definition 5.2.6 (Basic Flow Engine) *A basic flow engine* $\mathrm{ENG}(f, S) \mapsto [true, false]$ *can execute a flow instance $f = \mathcal{I}(\mathcal{F})$ of the flow model $\mathcal{F}$ given an event sequence $\mathcal{S}$. This execution results in a number of completed activities, and either in a successful execution ($true$) or unsuccessful execution ($false$).*

The event sequence S yields the necessary input for the flow instance and is "provided" over time by the CMS as individual event instances. The activities subscribe for their event type set as soon as they become active during the flow execution. They can then consume events from the event sequence S which may also be cached already in the flow engine event cache (cf.. Figure 4.1). When an activity a has received an event instance for each of its event type sets $E \in \epsilon_\alpha$ all outgoing transitions are evaluated to either true or false following Equation 5.6. Subsequent activities are then executed when all incoming transitions without a transition marker have been evaluated to true. If all have a marker at least one has to be true. Eventually most of the activities should be executed until no more events in S remain to be consumed or no further activities are ready for execution. In order to determine whether the execution of the flow instance has been successful, the following expression is evaluated.

$$\text{ENG}(f, S) = true \Leftrightarrow \forall a \in \Phi : \omega(a) = complete \quad \wedge \forall a \notin \Phi \omega(a) \neq active \tag{5.7}$$

Where $\Phi = \{a \in A : \phi(a) = true\}$ are all mandatory activities. This means, all mandatory activities in the flow must have been executed (in the HFM at least once) and no other activity is still allowed being executed i. e. violates the atomic property of the activity. Details of the execution result can be obtained from the completed activities an e. g. their data output. However this application output is not relevant in the scope of this thesis.

When processing uncertain context information, the flow engine must be parameterized with the navigation threshold t_n (cf. Equation 5.6) so that the conditions can be evaluated, leading to the execution of subsequent activities. If the stimulus provided by the full sequence S yields a state of the flow where the completion criteria is fulfilled (cf. Equation 5.7), the execution is successful ($true$). When the sequence S is wrong for the given flow instance, has errors regarding the event instances or the uncertainty prohibits a correct interpretation the flow execution fails ($false$). Please note that in this thesis, we do not cover failures caused by a combination of the given event sequence and another source, such as a failing service, as this is out of scope.

The basic flow engine ENG is capable to execute either basic flow models or the HFM defined later. As the mechanisms we propose in this thesis can be plugged into the basic flow engine we allow the flow engine ENG to be extended in the following way.

Definition 5.2.7 (Flow Engine Extensions) *A basic flow engine* ENG[] *can be extended with additional algorithms that FlowPal provides. These extensions are noted*

as generic algorithmic plugins $p \in P$, where $P = \{FlowCon, FlexCon, FEvA\}$ contains all algorithmic extensions of FlowPal.

For example, we can execute the flow f with the sequence S with the basic flow engine ENG(f, S) or using one with FlowCon ENG[FlowCon](f, S).

Each executed 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 5.2.8 (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 $\tau(\mathcal{T}, a, u) \mapsto e$ 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.

Definition 5.2.9 (Flow History) *A flow history $\mathcal{H}$ is an ordered sequence of flow traces $\mathcal{H} = (\mathcal{T}_1, \ldots, \mathcal{T}_i)$.*

Usually we refer to a flow history only in the context of a specific flow model $\mathcal{F}$ so that the history only contains traces of this model, although in practice it can contain all traces for different flow models.

5.2.2. Petri Net Similarity

To show that the basic imperative flow model is comparable to other workflow modeling languages such as BPEL, we sketch that it can also be mapped to to Petri Nets. A Petri Net (cf. [Rei10]) is a five-tuple $PN = (P_p, T_p, F_p, W_p, m_p)$ consisting of places P_p, transitions T_p, a flow relation $F_p \subseteq (P_p \times T_p) \cup (t_p \times P_p)$, a weighting function $W_p : F_p \to \mathbb{N}$ and an initial marking m_p. In order to map a flow (F) to a Petri Net, we would map the activities A to the places P_p, the transitions T to transitions T_p, the logical expression of each condition $c \in C$ would be mapped to an individual Petri Net and included for evaluation. As the flow model is missing an input similar to the initial marking m_p, we have to create it from the event sequence S but this also requires a new place $p \in P_p$ for each event type set, which is attached with a transition $t \in T_p$ to the place of its original activity a. The initial marker placement is then chosen that it starts the execution of the flow at its starting activity/place

and provides further markers for each of the events in the event sequence and their respective place in the Petri Net. Finally, we would have to define the flow relation F_p in such a way that it enables the activities similar to the condition evaluation and further enables the consumption of the markers from the initial placement when the respective activity becomes ready for execution.

5.3. Hybrid Flow Model Extensions

We extend the basic flow model to allow the modeling of more loosely structured flows, i. e. declarative flows describing constraints on the execution rather than an explicit execution path. It contains transitions as well as *constraints* between activities and, thus, is a mixture of classical imperative production workflows [LR00] and declarative flexible workflows [PSA07]. The transitions are annotated with boolean conditions over the possible set of context events, as defined earlier. *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 [AAH+09]. 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. In contrast, 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, such as person-centric flows. While the modeling paradigms on its own are already known, the way to compose them in a single coherent flow model is a novel contribution in this thesis.

Definition 5.3.1 (Hybrid Flow Model) *A hybrid flow model $\mathcal{F} := (A, T, C, L, \Phi, M)$ is a 6-tuple, consisting of a set of activities A, a set of transitions T, a set of conditions C, and a set of constraints L, where Φ is the set of mandatory activities and and M is the set of marked activities.*

The activities in hybrid flows are defined as before with the following additions. In a hybrid flow, activities may be executed arbitrary often and in any order. This means that an activity that meets all requirements again to change its state from *inactive* to *ready*, may also change its state from *complete* to *ready* again. The execution semantics are adjusted accordingly. A flow can complete its execution successfully when all mandatory activities have been executed at least once, i. e. are also in the state *complete*. Transitions and constraints limit this flexibility and impose structural ordering on the flow activities. While the transitions remain unchanged a constraint is defined in the following.

Definition 5.3.2 (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.*

The constraints are used only in hybrid flows. They can be grouped in different classes such as existence, (negative) relation, (negative) order [PSA07] and provided in a graphical representation (cf. Figure 5.2). At run-time they are converted to final state machines (FSM) [GH01] 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 completion criteria is then changed accordingly so that in addition to equation 5.7 it also fulfills the constraints. Φ again denotes the set of all mandatory activities.

$$\mathrm{ENG}(f,S) = true \Leftrightarrow \forall a \in \Phi : \omega(a) = complete \wedge \forall a \notin \Phi : \omega(a) \neq active \wedge \forall l \in L : l = valid \tag{5.8}$$

The definition of flow trace and flow history are suitable also for flows modeled with the HFM.

An example for and hybrid flow from the health-care scenario we studied (cf. Section 8.1) is depicted in figure 5.2. This example is only a fragment of a larger flow. Solid boxes depict mandatory *activities* that need to be executed unconditionally while dashed boxes are optional. For example, a_2 and a_3 are optional activities. The execution of a flow instance is valid if, one of the optional activities, both or non of them have been executed, while a_1, a_4, and a_6 need to be executed for the flow to complete successfully. 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 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. This constraint can be expressed as term in linear temporal logic, and this constraint belongs to

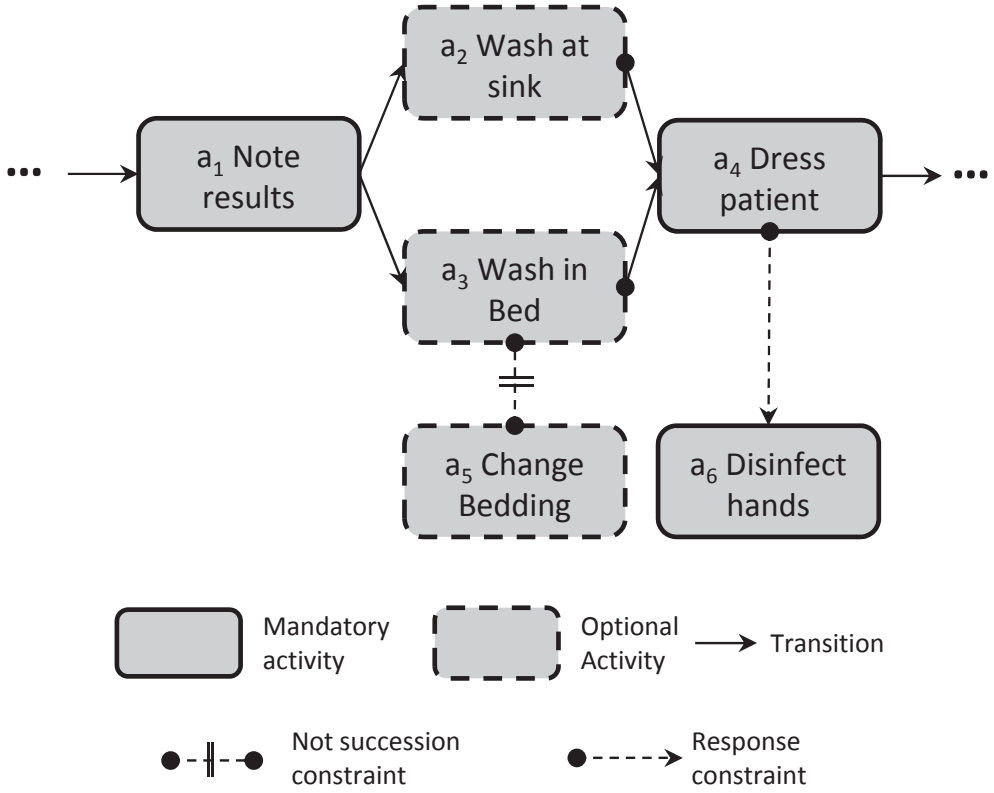

Figure 5.2.: Example workflow from a hospital scenario

the group of a negative relationship as the execution of one activity prohibits the execution of the other.

The overall semantics of the example are the following: When a caretaker arrives at this fragment, the results of preceding steps must be documented (a_1), 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 caretaker will not forget the results during the following steps. Then the caretaker has to take a decision: wash the patient at the sink (a_2) or in his bed (a_3), depending on the patients condition and mood. Based on the incoming context events the flow decides if either one or both are executed. If the caretaker decides to wash the patient in his bed (a_3), the bedding cannot be changed (a_5) since the patient never leaves the bed during the whole procedure. After the caretaker has completed the washing activity, the patient needs to be dressed (a_4). When finished, the caretaker must disinfect his hands (a_6) at some later point in time, possibly after a number of other intermediate activities. But while being executed, the flow allows also to execute the disinfect hands activity at any point in time. This is beneficial in two ways. First, the caretaker can flexibly decide to disinfect 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 caretaker to disinfect hands before he continues to care for another patient, this way enforcing the hospitals hygiene rules.

5.4. Discussion

We have seen that flows are becoming integrated into daily working routine in an increasing way. The basic imperative flow model we presented, is a suitable tool to describe processes with a well defined semantic and great focus on the integration of abstract context information. Compared to existing flow models that also incorporate context information, it represents mostly an alternative that eases the description of the following methods of FlowPal. But for a high degree of integration with humans, the flows must also be flexible enough to reflect the routine exhibited by different human stakeholder executing the same flow model. Our HFM is a novel original approach towards this flexibility as it supports directly the usage of both modeling paradigms in the same flow model. For standardized, yet critical tasks a hybrid flow provides strong guidance based on rigid structures that stem from imperative modeling (parts of the model which are imperative). But using the HFM, activities may also be structured by constraints that introduce a much loser coupling between them. These hybrid flows offer much more flexibility to the user. Along with this flexibility, they also potentially offer less information for FlowPal to improve flow navigation.

6. Probabilistic Reduction of Context Event Uncertainty

6.1. Preliminaries

In the previous chapter we have seen that a flow – given the right flexibility – is a suitable tool to support users in processes with rich human interaction. To achieve this with minimal explicit input, the flow system monitors what the user does in the real world in order to synchronize automatically with its actions and drive the flows execution forward respectively. This monitoring, which provides the majority of the input to the flow, is done by means of activity recognition (cf. [KL08, BKL10]). The respective data is provided to the flow by the CMS as context events. As we have discussed in Section 2.2.1, context events always suffer from uncertainty in their recognition especially when they get more abstract. Therefore, the flow must process and interpret the uncertain context events robustly, in order to be controlled effectively in an indirect fashion and complete its execution successfully.

The acceptance of pervasive applications depends on its capability to deal with these uncertain information in a transparent or at least graceful way. Events are be noisy with respect to their interpretation. This issue occurs irrespectively of the flexibility in the flow models [WHR10]. Further, due to the gained flexibility, it may not be completely clear which activity is actually executed next and thus awaits a respective event. Given the limits in time and cost to deploy a suitable context recognition system, this is a very challenging problem.

We have also discussed in detail the various approaches that include context information in workflows in Section 5.1.2. Please note again, that none of the approaches considers the uncertainty of context events when processing context information. One of the factors that can aid overcoming this limitations, is structured application knowledge. A flow provides information about its target domain and application, the activities to perform, and their temporal ordering. This information, which we introduced as flow knowledge, is encoded in the flow and can be used to improve application quality and adaptability.

⋆Con (StarCon) uses flow knowledge to decrease the uncertainty of single context events. This leads to an improved accuracy of the recognition by the flow engine and hence to a more robust execution of flows given uncertain context information.

In the next section we discuss related approaches, before we introduce the common algorithmic principle of ⋆Con in Section 6.3. After that, we introduce both its algorithmic variants, FlowCon in Section 6.4 and FlexCon in 6.5. Subsequently we study the influence of the uncertainty model on ⋆Con in Section 6.6, where we introduce HyperFlowCon.

6.2. Related Work

A combination of uncertain context information and workflows has found little to no attention in research, yet. Furthermore, we already discussed abstract context recognition (cf. Section 2.2.1) and context integration in workflows (cf. Section 5.1.2). But, there is one area which is basically related to what we achieve with ⋆Con. In order to create a flow either a domain expert creates the process model or a technique called *workflow mining* is applied recorded data to extract the workflow from the observations made. This workflow mining is also used as part of our evaluation strategy and we will cover more details on the concrete case study in Chapter 8. In general, a flow mining algorithm performs an analysis on certain event logs that record the execution of activities, which should be covered by the created flow model. Without knowing the structure of a flow model, a flow trace $\mathcal{T}$ is a good representation for such an event log.

Workflow mining comes in two different flavors. On the one hand, a new workflow is created from event logs [DMV+05] of different applications in order to visualize the actual flow of work and automate the created workflow using classical workflow management techniques. On the other hand, existing workflows can be used to extract knowledge. While we extract also knowledge from flows, there are no approaches that improve context processing this way. Buffett and Geng [BG09] 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 (BNs), resolves the ordering of activities in the generated workflows and analyzes the paths taken in workflow execution. However the algorithm is applied to already collected event logs and the workflows are mined with an offline algorithm. Furthermore, it assumes that the log data contains no uncertain information, i. e. no real-world context information is taken into account. In contrast, we know the flow structure and learn the event dependencies at run-time, also taking uncertain context information into account.

6.3. ⋆Con Algorithmic Principle

The goal of ⋆Con (StarCon) is to decrease the uncertainty of an event instance e. This means, 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. We already classified this information as flow knowledge. As the human user in charge of executing the flow is aware of the flow and tries to complete his work successfully, we can assume that the context information generated in the real world depends on the current flow state.

Let us assume that the flow engine has started the execution of an activity a_1, and receives the events $\epsilon_\alpha(a_1)$ of the types associated with a_1, including E_α (cf. Section 3.2.1). In a system without ⋆Con, the flow engine would simply compare the probability $p = I_{E_\alpha}^e(u)$ with the engine's *navigation threshold* t_n and execute the respective transition if $p \geq t_n$. This simple approach is depicted in Figure 6.1 on the left (also cf. Section 5.2). In contrast, ⋆Con 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 interpretation $I_{E_\alpha}^e$. This interpretation represents a probability distribution in the basic case. The results of this interpretation are then used again for the threshold comparison, leading to more reliable decisions and, thus, a more robust flow navigation.

⋆Con uses Bayesian Network (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 context events,

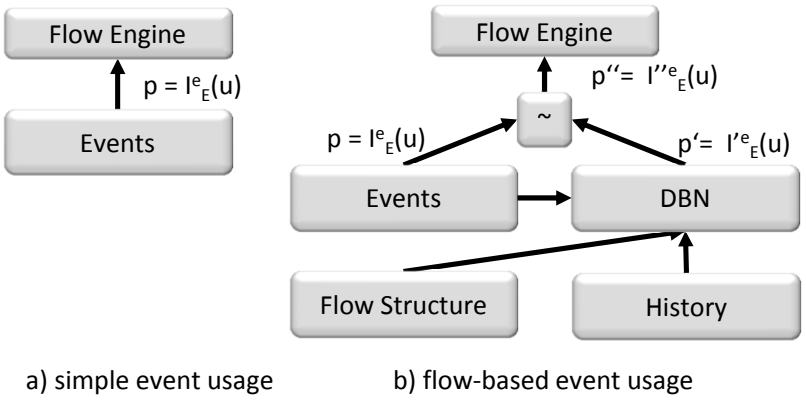

Figure 6.1.: Architecture overview

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 from the flow history of previously executed flows. This is shown in Figure 6.1 on the lower right.

When a flow instance is executed, every incoming context event e is sent to the BN. Any such event is associated with an interpretation/probability distribution I_E^e (cf. Definition 3.2.3). The BN infers an *additional* conditional interpretation/probability distribution $I_E^{\prime e}$ for e over E. I_E^e given by the CMS and $I_E^{\prime e}$ given by the BN are combined, yielding an overall interpretation $I_{E_\alpha}^{\prime\prime e}$, which is then used by the flow engine to make its navigation decision. Our evaluations show that if $e \in u$ then, on average, $I_E^{\prime\prime e}(u) > I_E^e(u)$. Hence, ⋆Con reduces the uncertainty contained in the original distribution such that the flow engine can make correct threshold decisions more often.

We have applied the principle of ⋆Con (StarCon) in two different variants. The first one, Flow Context (System) (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. The details of FlowCon are described in the next section. In contrast to this, the second variant **FlC!** (**FlC!**) 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 Network (DBN) that can handle dynamically changing dependencies between context events. Using exact inference to get $I_E^{\prime e}$ from a complex DBN, is computationally infeasible. Therefore, we use an approach based on *particle filters* [RN02] 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 6.5.4.

6.4. FlowCon

In this section we describe the details of FlowCon algorithm. In order to use FlowCon the following requirements have to be met. The application must be specified as a flow model that is highly structured and uses the basic imperative flow model that we introduced in Section 5.2. Further, this flow model must remain static, in order to collect flow traces of the execution of this flow model. The traces are used to train FlowCon and when the flow model changes this training should be started again.

A reasonable example for this type of flow based application can also be found as part of the morning hygiene example we already mentioned in Section 5.3. But

in order to limit the flexibility of the application, we focus on a blood sample examination process. This process can be modeled using the basic imperative flow model only, which is the type of flow FlowCon is capable to handle. 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 respective caretaker executing the flow.

The caretaker performs a blood sample examination when it is scheduled for a patient. This is documented in the patient record. 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 6.2. To obtain a blood sample the caretaker 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 caretaker basically has two variation options. She can either disinfect the elbow pit first and then unpack the butterfly or the other way around. However she must complete both activities before she can set the butterfly. Moreover, she is free to choose the order in which the individual samples are taken after she has set the butterfly.

These 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 caretaker executes. Because of this it is much harder to recognize the context information for the following activity compared to a scenario with a predefined fixed sequence of activities. Each context may appear and from the point of the CMS there is no way to tell which one is more likely. In this case, FlowCon is able to increase context recognition performance, because it can learn this from flow knowledge. Furthermore, when the flow execution waits for an

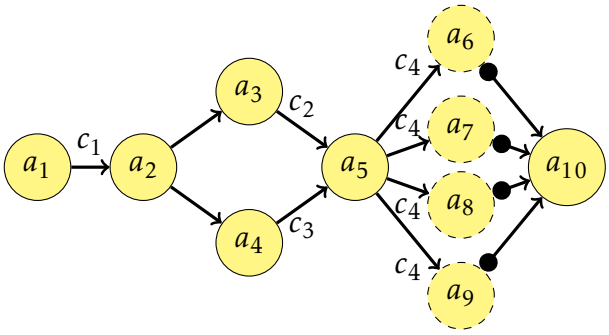

Figure 6.2.: Blood Sample Flow

activity such as 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 caretaker 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. FlowCon aims to increase the accuracy of the context events the flow execution relies on. More specifically, we adjust the probability distribution given by the interpretation of an event instance I_E^e, so that the statistically most probable event will be favored. The probability of this event is increased, while simultaneously the probabilities of the possible other events are decreased. This way the accuracy of the events expected by the application will be increased and their uncertainty is decreased. We train a Bayesian Network (BN) [RN02] to extract the information from the flow and assess which event currently is the most probable. The training of the BN consists of two phases: structure learning and parameter learning. While parameter learning can be achieved efficiently from a set of given observations (flow traces), learning the optimal structure of a BN from such data is NP-hard. FlowCon 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. The structure of the BN is reusable as long as the flow remains unchanged. In the second step, it 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 instance, it is possible to perform both online. The creation of the BN structure can be computed fast from the given flow model. Training the parameters online can further make use of the newest available flow traces and hence favor a changed human routine in fast fashion.

The third step – the information combination – must executed at run-time (cf. Figure 6.1. Usually the application has to deal with the provided probability of an event (e_1, p), but FlowCon queries the BN for the current event, using the actual execution state of the flow and the already received event instances. It then combines the event with its derived statistical probability from the BN (e_1, p') and generates a new probability for the event (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.

6.4.1. Flow Structure Analysis

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 (cf. Figure 6.2), the occurrence of the event e_1—apply cuff—necessitates the occurrence of the event 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 events of a_2 and the events of a_3, a_4. A single caretaker could disinfect the arm first every time and then unpack the butterfly. This does of course change the statistical dependency between the events. 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 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, this way dealing with changing behavior of caretakers.

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. Each node n of the BN represents a discrete random variable (RV). The state space of a node n is equal to the event type E adding a `nu11` class. For every event type of an activity we create a node $n = a.E$ identified by the name of the activity and the event type. After that, we add the dependencies as arcs between those nodes where the activities also have a directed dependency, or more formally:

$$((a_x, a_y) \in T) \wedge (\exists i : \epsilon_{a_x}(i) = E_x) \wedge (\exists j : \epsilon_{a_y}(j) = E_y) \Rightarrow ((a_x.E_x), (a_y.E_y) \in D) \tag{6.1}$$

Algorithm 6.1 shows the pseudo code for creating the BN structure from the flow. First, we create all the nodes by assigning them the combined identifier of activity and event type they refer to (Lines 2-8). After that, we create directed dependencies between those nodes where the activities also have a directed dependency (Lines 10-16). This completes the structure learning of the BN.

Please note that the structure learning has been very simple – effectively only a mapping – because we have the flow structure which provides us with a realistic assumption which events are related to each other. Usually, learning the optimal structure of a BN from a set of observed values is NP-hard and suitable heuristics need a large training set to perform well. 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 (with the same structure) in use for different locations where the flow is actually deployed and executed, or for different actors that are associated with the flow (having a dif-

Algorithm 6.1 Structure Analysis

```
    N ⇐ {}
 2: for each a ∈ A do
      IdPrefix ⇐ a.ID
 4:   for each E_i ∈ ε_a do
        IdSuffix ⇐ E_i.ID
 6:     N ⇐ N ∪ createBNNode(IdPrefix + IdSuffix)
      end for
 8: end for
    D ⇐ {}
10: for each t ∈ T do
      for each E_i ∈ t.a_x do
12:     for each E_j ∈ t.a_y do
          D ⇐ D ∪ createArc(a_x.ID + E_i.ID, a_y.ID + E_j.ID)
14:     end for
      end for
16: end for
```

ferent set of learned parameters). For example, there could be a single BN trained for every caretaker to take personal habits into account when executing the flow and processing the context information.

Usually, the number of events grouped together in an event type is rather small. A practical example for an event type may be E_{alpha} which groups together different modes of locomotion, as defined in Section 3.2.1. The size of the event type is important because the number of entries in the conditional probability table (CPT) of a BN grows exponentially with the number of values a single node can have, and the number of dependencies. So if there would be a large number of events grouped in an event type and there would be multiple dependencies between event types of similar size, the space necessary to represent the BN may become to large. However, for the observed events in the traces and practical considerations, BN are suitable to represent the dependency structure.

6.4.2. Bayesian Network Training

The next step is to train the BN with the actual statistical dependencies that have occurred. At first the CPTs of each node are initialized with an uniform distribution. Then, we use a common maximum likelihood estimator with a uniform prior and update the values in the BN (cf. [BFH+09] p. 123). Therefore, we need a training data set with observations for the value of each node. The flow traces that are collected for each execution of a flow are ideally suited as a data set for training. As

the event instances are stored together with the flow trace, it can be converted to an observation for each event represented by a node in the BN.

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 `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. 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.

6.4.3. Runtime Information Combination with Bayesian Networks

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 Flow instance that is currently executed, are the past events that have already been recognized. We take all available context events as evidence into account.

When the next event instance I_E arrives, we compute the conditional probability for this event using the previous event instances as current observations. This means we can use the executed trace of the flow as evidence and hence, reduce the search space in the BN. This also effectively counters the problem having very small probabilities after a number of inference steps. The result from the BN is also a probability distribution function $I_E'^e$ for the event type E. We combine this result with the probability distribution function I_E^e of the actual event instance e. We compute the average probability for each event in the distribution and normalize the results afterwards to make the probability distribution valid again.

$$
\begin{aligned}
I''^{e}_{E}(u) &= \frac{I'^{e}_{E}(u) + I^{e}_{E}(u)}{\mathcal{I}} \\
\mathcal{I} &= \sum_{u \in E} I'^{e}_{E}(u) + I^{e}_{E}(u)
\end{aligned}
\tag{6.2}
$$

This combination adjusts the probability of each event by the amount of its statistical probability under the given context of the previously occurred events. We used this method for FlowCon at first due to the lack of a better intuition of what to do with the results from the BN. As we will show later in Section 6.6, a much more meaningful combination of information can be achieved, when we change the representation of uncertainty of the context events. Here we actually adjust the context measurement. The probability of an event that occurs more often is increased, while the other way around, its probability gets lowered. Note that higher probability also means a higher accuracy for the measured event. 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 behavior.

6.5. FlexCon

FlexCon is targeted for flows modeled with the Hybrid Flow Model (HFM), suitable for more flexible scenarios. It uses the same algorithmic principle but has to deal with more flexibility when executing flows. But this forces us to rethink also on the use of BNs to represent the statistical dependencies between context events. As the HFM allows the repeated execution of activities in flows, BNs are no longer possible. If activities are repeated there would be additional nodes required in the BN for each execution. But it remains unclear until the concrete execution of the flow instance how often an activity is executed. Further, in case the order of execution is left to the user it may be unclear, which activity is conditionally dependent on the other. This may also change form instance to instance. As there may be statistical dependencies in both ways this would cause a cycle to occur in the structure of the BN as constructed by FlowCon (cf. Algorithm 6.1). For example in the flow shown in Figure 5.2 the activities a_3 and a_5 are tied together by a not succession constraint. But while a_3 is embedded strongly with other activities, a_5 may be executed either before or after a_3. So we cannot assign a directed conditional dependency between the two activities, without introducing a cycle in the BN. But a cycle invalidates the BN structure and would make the result useless. To

tackle this issue we have to take the execution time of an activity as an additional factor into account.

FlexCon uses *Dynamic Bayesian Network (DBN)s* to interpret context events depending on the current state of the flow. A DBN 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. It also has a time dimension and each time slice does only take into account a part of the the flows execution state. Its structure is basically identical the BN of FlowCon but unrolled over time. FlexCon builds the structure of the DBN from the flow model and trains the DBN using traces of previously executed flows. This is shown in Figure 6.1 on the lower right. We explain the details of the construction algorithm in the next section.

When a flow instance is executed, every incoming context event e is sent to the DBN. Any such event is associated with a probability distribution I_E^e (cf. Definition 3.2.3). The DBN 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 DBN are combined, yielding an overall distribution $I_{E_\alpha}''^e$ 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, FlexCon reduces the uncertainty contained in the original distribution such that the flow engine can make more correct threshold decisions.

Using exact inference to get $I_E'^e$ from a complex DBN, such as the one built from the flow model with an online algorithm, is computationally infeasible [Mur02], as its complexity has a lower bound of $\Omega(K^{m+n+1})$. Here, any event type set has at most K discrete values, m may in worst case be the sum of all event type sets used in the flow model and n is the number of parent nodes for the inference in question. Therefore, we use an approach based on *particle filters* [RN02] 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 6.5.4.

6.5.1. 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 $\{X_1, ...X_n\} = \bar{X}$, where $\bar{X}$ represents all the nodes. The edges $d \in D \subseteq \bar{X} \times \bar{X}$ of the BN define a conditional dependency from the source random variable (RV) to the target RV. In FlowCon, we used BNs as the flows were based on imperative models that specify the complete execution order. Therefore, the simple static BNs were sufficient. However,

the HFM for FlexCon allows the user more freedom to execute activities. There can be more than one connected component in the flow model and the execution order of some activities is only defined by constraints. Further the user is also allowed to exectue some activities more than once. The static BN model cannot represent this kind of user behavior as it would introduce cycles in the structure. Therefore, FlexCon employs DBNs which are tailored for dynamically changing systems.

In a DBN [RN02, Mur02], the values of the RVs changes over time. Further, the observed values for the RVs in the current *time slice* $\bar{X}_t$ can 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:

Definition 6.5.1 (Dynamic Bayesian Network) *A* $\mathcal{DBN} = (\bar{X}, \mathcal{TM}, \mathcal{PD})$[1] *consists of a set of random variables* $\bar{X}$*, a transition model* $\mathcal{TM}$ *and an initial prior distribution* $\mathcal{PD}$*.*

The structure of the transition model $\mathcal{TM}$ must be derived from the available HFM and its parameters must be tuned according the available flow history, similar to FlowCon. Further we must also provide suitable values for prior distribution $\mathcal{PD}$ in order to start with the reasoning in the DBN when the flow execution starts. In the following we describe the algorithm FlexCon uses to construct each of these components.

6.5.2. Flow Analysis and DBN Construction

FlexCon creates the random variables $\bar{X}$ in the DBN based on the event type sets and activities of the corresponding flow. Let $\mathcal{F}_1 = (A, T, C, L, \Phi, M)$ be the flow model from our example in Section 5.3. 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. Each RV $X = \chi(a, E)$ 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 $E_\alpha \in \epsilon_{a_1}$ (cf. Section 3.2.1). The respective random variable $\chi(a_1, E_\alpha) = X_\alpha$ can assume any value from $\{$*person walking*, *person sitting*, *person standing*, $\perp\}$. $\chi(a, E)_t$ (the RV of the event type set E assigned to activity a) and $\bar{\chi}(a, \epsilon_a)_t$ (all event type sets for activity a) refer to

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

the respective time slice t. In order to construct the set $\bar{X}$ of random variables for the $\mathcal{DBN}$ we use the algorithm depicted in Algorithm 6.2.

Algorithm 6.2 DBN Node Mapping

```
    X̄ ⇐ {}
 2: for each a ∈ A do
      IdPrefix ⇐ a.ID
 4:   for each E_i ∈ ϵ_a do
        IdSuffix ⇐ E_i.ID
 6:     X̄ ⇐ N∪ createDBNNode(IdPrefix + IdSuffix)
      end for
 8: end for
```

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 (i. e. 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 these order relations to the transition model, introducing directed edges (dependencies) from one time slice to the next. The strength of these dependencies is learned from flow traces 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 5.2: 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 $l_1 = \Box(a_3 \rightarrow \neg(\Diamond(a_5)))$ (cf. Definiton 5.3.2)represent the *not-succession* constraint in the example in Figure 5.2. 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 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 (cf. Section 5.3), 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 5.2, where the execution of a_6 has absolutely no dependency on the execution of a_4. Algorithm 6.3 shows the mapping of the constraints more formally, where Σ_l denotes the alphabet, Z_l the set of states and $\overset{a}{\Rightarrow}$ the respective transition relation given the activity $a \in \Sigma_l$ for the final state machine for constraint l. Further $permanentlyViolated$ is a predicate that becomes true if the given state z denotes that the respective constraint l has become permanently violated.

6.5.3. Dynamic Bayesian Network Training

In order to learn the strength of dependencies in the DBN, we use the flow history as training data, counting the occurrences of all context event pairs and learning their joint probability distribution. Again the flow trace acts as a suitable source for the required evidence and again we use a maximum likelihood estimator for the training [BFH+09]. 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. We determined this size experimentally, and found that 25 to 50 flows are sufficient to provide a stable behavior of the DBN. This helps controlling the effectiveness of the learning procedure when facing a changing behavior of the flow system.

For dependencies originating from flow transitions, the simple counting of context events and computing their relative frequencies as described at the beginning of this section 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 context 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.

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

Algorithm 6.3 DBN Transition Model Mapping

$\mathcal{TM} \Leftarrow \{\}$
for each $t \in T$ **do**
  for each $E_i \in t.a_x$ **do**
    for each $E_j \in t.a_y$ **do**
      $\mathcal{TM} \Leftarrow \mathcal{TM} \cup$ createArc$((a_x.ID + E_i.ID)_t, (a_y.ID + E_j.ID)_{t+1})$
    end for
  end for
end for
for each $l \in L$ **do**
  for each $a_x \in Sigma_l$ **do**
    for each $a_y \in Sigma_l$ **do**
      if $\forall z \in Z_l : Z \overset{a_x}{\Rightarrow} Z$ **then**
        $skip$
      end if
      if $\exists z \in Z_l : z \overset{a_x}{\Rightarrow} z' \wedge permantnlyViolated(z')$ **then**
        for each $E_i \in a_x$ **do**
          for each $E_j \in a_y$ **do**
            $\mathcal{TM} \Leftarrow \mathcal{TM} \cup$ createNegArc$((a_x.ID + E_i.ID)_t, (a_y.ID + E_j.ID)_{t+1})$
          end for
        end for
      else
        for each $E_i \in a_x$ **do**
          for each $E_j \in a_y$ **do**
            $\mathcal{TM} \Leftarrow \mathcal{TM} \cup$ createArc$((a_x.ID + E_i.ID)_t, (a_y.ID + E_j.ID)_{t+1})$
          end for
        end for
      end if
    end for
  end for
end for

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: $P(\textit{person walking}) = 0.05$, $P(\textit{person standing}) = 0.18$, $P(\textit{person sitting}) = 0.65$, $P(\textit{person disinfect}) = 0.01$ and $P(\perp) = 0.12$. In most of the cases the caretaker is sitting while writing the results. In some cases, they do it while standing. At other times, there was no meaningful evidence at all ($\perp$). The rates for the uncommon mode of locomotion (*person walking*) is even lower. Please note that these are the results observed by the flow application regarding previously executed flow instances, not the actual readings from the CMS for E_α. The result are individ-

ual probability distributions for each of the past starting activities for this flow model.

6.5.4. Clustered Particle Filtering with Dynamic Bayesian Networks

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 RVs (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 RVs in time slice $t + 1$. Exact inference is infeasible for complex DBNs like the ones generated by FlexCon. 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. FlexCon uses a heuristic approach that is based on particle filters [RN02]. Therefore, 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 single particle is a sample taken at random from the distribution of the DBN at a certain time t. A particle filter approximates the exact joint probability distribution by generating a set of particles $N(\bar{X})$ for all random variables $\bar{X}$. 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 [RN02].

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. [NPP02]. Algorithm 6.4 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.

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.

Algorithm 6.4 Clustered Particle Filter Algorithm

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

For example, consider the trace $\mathcal{T}_1 = (a_1, a_6, a_3, a_4, a_6)$ for the process depicted in Figure 5.2. 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. a_6 is executed next, relying on the prior distribution (and thus the particles) from $\chi(a_6, \epsilon_{a_6})_1$. While the particles $\chi(a_6, \epsilon_{a_6})_1$ are propagated normally, all other particles $N(\bar{x})_1 \setminus \chi(a_6, \epsilon_{a_6})_1$ are propagated to time slice $t = 2$, 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 $N(\bar{\chi}(a, \epsilon_a)_t)$ i. e. evidence from other activities. As a consequence, we introduce an error in the inference. When there is a conditional dependency between $\bar{\chi}(a', \epsilon_{a'})_{t+1}$ and any $X \in (\bar{X} \setminus \bar{\chi}(a, \epsilon_a))_t$, this evidence can not be taken into account. However, the majority of $X \in (\bar{X} \setminus \bar{\chi}(a, \epsilon_a))_t$ will be independent from the variables in $\bar{\chi}(a', \epsilon_{a'})$, because there is no dependency defined by the flow. Furthermore, at time $t + 1$, we have an actual observation available – the navigation decision of

the flow – and will use this evidence for further inference when weighting the particles (cf. line 7). Therefore, the introduced error will be compensated in the next time slice and we actually discuss in Section 9.1 that not using this evidence makes FlexCon a bit more robust. To avoid this problem, we could also *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, as this distribution only represents the global state of the particle filter, but not real evidence.

After the propagation phase (Line 10), 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 $I_E^e(u)$ (Line 7). Based on the computed weights all the particles for $\bar{\chi}(a, \epsilon_a)$ are resampled according to the distribution of the weighted particles (Line 8).

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 they are weighted for the next resampling. 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. With p' the DBN provides us again with a second source of information similar to FlowCon. This information can then be combined with the actual result from the received event instance. In the basic case, just averaging between the two sources of information, all probabilities $p = I_E^e(u)$ are added to the respective p' and the resulting distribution is normalized again, yielding $I_E''^e(u)$ similar to FlowCon (cf. Equation 6.2). After the event instance has been modified it is processed normally by the CE and the flow engine.

6.6. Uncertainty Model Variants

Up to now, we used ⋆Con in both its variants to alter the interpretation I_E^e of a received context event. We derived a second interpretation form ⋆Con and combined it with the context event assuming both contributing the same amount of information (cf. Figure 4.2). But so far, we only used a basic version of the Condition Evaluator (CE) using probabilistic logic to reason on the resulting interpretation $I_E''^e$ (cf. Definition 5.2.4). Basically the CE provides two features. First, it checks the final interpretation of the event for the desired context information. Second, it combines this context according to the evaluated condition using the appropriate

operators (i. e. and $\wedge$, or $\vee$, not$\neg$). But this combination may suffer from greatly reduced probabilities, when a complex condition with multiple and operators is used. As a result even with very accurate results from the CMS (and/or ⋆Con) the flow engine may have problems when evaluating the condition against the navigation threshold (cf. Equation 5.6).

We now investigate two alternative approaches, aiming for a more suitable representation of uncertain context information for the flow so that the processing of context events becomes more robust. On the on hand, we want to combine the information from ⋆Con with the real evidence in a more meaningful way than just averaging between the two. On the other hand, we want to make the reasoning process better using stable operators for reasoning on uncertain context information that do not increase uncertainty again during condition evaluation. The reasoning could in theory be used on its own to evaluate the context conditions but the combination requires the use of either variant of ⋆Con in order to have a second source of information. We investigated two alternative approaches and both complement the use of ⋆Con in any combination. The first approach is introduced in the next section and covers the integration of Subjective Logic (SL) in the Condition Evaluator (CE) as well as the representation of uncertainty. The second approach investigated the use of four-valued logic (4V) as suitable replacement for Probability Theory (PT) to reason on the conditions and is presented in Section 6.6.5.

6.6.1. Subjective Logic Condition Evaluator

The novel Subjective Logic (SL) Condition Evaluator (CE) [WHR13] is a sophisticated method for combining the two sources of information discussed above: the probability distribution I_E^e measured by the CMS for the real-world event instance and the probability distribution $I_E'^e$ inferred for the same context event from ⋆Con. In our original approach we just averaged between the I_E^e and $I_E'^e$, thus using a fixed amount of 50% from each information source (cf. Equation 6.2). In order to combine both sources in a meaningful way, we propose the use of Subjective Logic (SL) (cf. [Jøs11]) for two reasons. First, we can dynamically adapt and control the amount of information we use from I_E^e and $I_E'^e$, based on the amount of ignorance attributed to the incoming context event. Second, it allows us also to reason on the result of a condition using 2nd order probabilities. SL is a type of probabilistic logic taking uncertainty on probabilities and incomplete knowledge into account. A belief in a certain situation is expressed as so called *subjective opinion*, that describes the subjective belief of an observer for a given proposition. In general a subjective opinion is composed of *belief*, *disbelief* and *ignorance*. Belief is the evidence that supports the proposition, disbelief is evidence that opposes the propositions and ignorance represents the observers uncertainty on the

evidence given available for the proposition. We will use this representation to describe the uncertainty of the context events and adapt the way we combine the its information with the one from ⋆Con. As we will present in the evaluation this approach leads to a reduced dependency on ⋆Con giving more weight to the information from context event. Further, we make some modifications to add a minimal amount of ignorance when combining both information sources and render the actual navigation threshold adaptive to the current event to increase the robustness of the navigation decisions even further.

We will start introducing the basic principles we used from SL and how they are integrated into SL CE. Following that, we discuss the details of the minimal amount of ignorance and the navigation threshold adaptation. In Sections 9.2 and 9.3, we will discuss our evaluation results.

6.6.2. Integrating SL in the CE

In order to apply SL with the CE, we first have to extend the definition of the event instance and our model of uncertainty with terms for *belief*, *ignorance* and a *base rate*. Then we can construct *subjective opinions* and reason on them.

In terms of SL, an event type u represents a single atomic event in the real-world for a given situation and the event type set represents our frame of discernment $\theta = E$. The belief mass of a subjective opinion is assigned to a reduced power set of θ that we refer to as $\mathcal{R}(E) = 2^E \setminus \{E, \emptyset\}$. The full opinion E is not a valid subset because this value represents full ignorance and hence we have no belief in this situation. The $\emptyset$ is not regarded as well, as E is assumed complete i. e. it already contains a *null* class.

Given $\mathcal{R}(E)$ we can now define any basic belief assignment (bba) (cf. Equation 2.2.1) for E as $\vec{b}_E$ that distributes the belief mass over the sets of $\mathcal{R}(E)$. We could use the distribution of the event instance $I_E^e(u)$ and map the probabilities to an bba for an event instance e. However this would lead to so called dogmatic beliefs as a probability distribution is additive to 1 (there is no ignorance). In order to add an amount of *ignorance* we use a simple approach that is based on the *significance* $\mathcal{S} = \hat{u} - \frac{1}{|E|}$ of the current event instance, where $\hat{u} = \arg\max_{u \in E} I_E^e(u)$. I. e., the higher the largest probability $\hat{u}$ in I_E^e deviates from the uniform distribution probability $\frac{1}{|E|}$, the higher the significance of the event e. This definition assumes that a large amount of probability for a single event type indicates a valid measurement. However, if all values in the distribution are close to a uniform distribution, thus the significance of the event instance is low, we assume a higher amount of ignorance. We reflect this when creating $\vec{b}_E$ by multiplying each probability with $\hat{u}$. The remaining mass $1 - \hat{u}$ is then assigned as ignorance i_E. Please note that this transformation step could also be defined for CMS that provides less information.

For example, if the CMS gives an event instance containing $e = (u, E, p)$ where p is the probability of event type u, which is part of the event type set E. Then we could assign this probability as belief to $\{u\} \in \mathcal{R}(E)$ and keep the remaining belief $1 - p$ as ignorance i_E. As long as this kind of transformation can be well defined SLCE can support basically any kind of CMS interface.

Finally, we introduce the *base rate* that expresses the observers expectation for an event type. If no information on this expectation is available, we have to use a uniform expectation value for any element of $\mathcal{R}(E)$. However if we have some source of information how to handle uncertainty we can assign use the base rate to project the uncertainty back to a probability based using the base rate. ⋆Con provides us with exactly this biased expectation and allows a meaningful and practical assignment of base rates. Let $\vec{a}_E(u)$ denote the base rate for event type u for a given event instance. Then we assign $I'^e_E(u)$ as base rate $\vec{a}_E(u)$ for this u.

Again, note that the amount of ignorance i_E directly determines the amount of the base rate used to reason on this situation. I. e., the higher the ignorance, the more we use information from ⋆Con, because the reading from the CMS contained less useful information and vice versa. If the ignorance is low, we can rely on the measured event from the CMS and have to use less information from ⋆Con. This is of huge importance to applying ⋆Con as our evaluations in Chapter 9 will show. Using this representation we can balance in an adaptive and fine grained manner what source of information to trust more in order successfully execute the flow. When we have a very accurate context measurement we can use ⋆Con just as a validation tool whether the received context event actually fits in the overall statistics of the executed flow. Given a very inaccurate CMS or just having a temporary degradation in context event accuracy ⋆Con will take over most of the responsibility an provide more information to aid in successful flow execution.

Having all the individual components we can now construct the subjective opinion ω^A_X, where A is the observer and X is the frame of discernment. There exist three different classes of subjective opinions, *binomial*, *multinomial* and *hyper-opinions*. The latter are the most general, but we only need the simple *binomial opinions*. They are restricted to frames of discernment where $|X| = 2$. At first this seems to contradict with event type sets with more than one element. However, according to Definition 5.2.4, we always check against one single event type. Therefore we can create a binomial subjective opinion $\omega^{*Con}_u = (b, d, i, a)$, with the observer ⋆Con and using the basic belief assignment $\vec{b_E}$, the ignorance i_E and the base rate $\vec{a}_E$. As we have a binomial opinion we only take the information into account that this event represents u or not. The values are for the binomial opinion are computed as follows. The belief in the opinion is just its value from the belief vector $b = \vec{b_E}.u$, that has been created from I^e_E using $\hat{u}$. The disbelief is all contradicting evidence

we have in the belief vector $d = (\sum_i \vec{b_E}.i) - b$. The ignorance is given by the remaining probability mass $i = i_E$ and finally the base rate is given by the probability of ⋆Con on the event type u as follows $a = I_E'^e(u)$.

As the basic expressions of the condition are now binomial subjective opinions we change the way our basic CE works accordingly. Instead of using probabilistic logic to assess the condition we apply their SL counterparts. Only before we check against the navigation threshold t_n, we project the final opinion back to probability space.

6.6.3. Minimal event ignorance

We mentioned before that FlowPal used averaging to combine incoming event instances with information from ⋆Con. By applying SLCE we can limit the amount of information used from ⋆Con depending on the significance $\mathcal{S}$ of the event instance. Our first experiments with SLCE showed two things. First, in general the ignorance i_E is less than 0.5, thus we use less information from ⋆Con than in the original approach. This is beneficial because we actually rely more on real input than on its statistical relevance. Second, when FlowPal with SLCE is exposed to a context system that provides wrong information with a high significance, i. e. $\hat{u} \geq 0.5$ but for the wrong event, FlowPal with SLCE is prone to believe this wrong information more easily. The high significance leads to a lower usage of ⋆Con than compared to the version without SLCE, and thus provides less help detecting this wrong information. This did not happen before due to the fixed averaging between I_E^e and $I_E'^e$. In general, we have observed a lower usage of ⋆Con in FlowPal with SLCE. Therefore, in order to counter the mentioned wrong behavior, we artificially increased the amount of uncertainty for an event instance to double check it in every case with ⋆Con to some degree. We adjusted i_E such that the uncertainty is always above the *minimal event ignorance* $\check{i}$, i. e. the ignorance in the binomial opinion $i = max(i_E, \check{i})$ is either the minimal event ignorance or the ignorance given by the event instance, whichever is greater. As a result we still use less information from ⋆Con in most of the cases but still have a fair chance to detect wrong events, that have a high significance. In general, the more likely the CMS provides high significant wrong events, the higher $\check{i}$ should be. For our experiment setup a value of $\check{i} = 0.4$ has been optimal. When deploying a real-world system it would also be possible to monitor the performance of the CMS over time and develop a meta-heuristic that adapts $\check{i}$ based on the observations.

6.6.4. Adaptive Navigation Threshold

Having applied the minimal event ignorance $\check{i}$, we noticed that the flows failed more often when processing an event instance with $|E| > 3$, because the threshold of important conditions could not be met despite a fairly accurate context event. This effect was due to the still static navigation threshold $t_n = 0.4$. This threshold has been introduced in previous work [WHR09]. It is tailored for the scenario and the average recognition probabilities we achieved in the hospital scenario. However, it does not take the size of the event type set into account. For $|E| = 2$ a probability of 0.4 represents a rather low significance (expectation value = 0.5) , while for $|E| = 4$ (expectation value = 0.25) a significance of 0.4 is harder to achieve. Therefore, we adapt the threshold to the size of the event type set in question for each condition evaluation as the conditions are automatically generated in our simulations (cf. 8.2). We started with a threshold that is equal to the expected value of the given event type set and added a certainty margin. In general the certainty margin should be higher the better and more reliable the CMS provides the correct context recognition and vice versa. In the experiments that we conducted 28% have been determined as optimal value for the certainty margin. As for the minimal event uncertainty $\check{i}$ given enough observation of the CMS performance in a real-world deployment a meta-heuristic adapting certainty margin becomes feasible.

6.6.5. Four-Valued Logic Condition Evaluator

In a second approach we investigated the use of four-valued logic as a replacement to standard PT. The four-valued logic (4V) [BJ77] is founded on the basic principle that queries are send to knowledge database. As this kind of databases tend to contain incomplete and also contradicting information, there are more meaningful answers to a query of that database system than "true" (t) and "false" (f) but also ignorance $\bot$ (no knowledge) and contradiction $\top$ (evidence for true and false). M. L. Ginsberg [Gin88, Gin90] generalized 4V and introduced the concept of a bilattice in conjunction with two partial orderings. The first one is $\leq_k$, the so called knowledge order, that allows to compare values in the bilattice against their knowledge level. The second one $\leq_t$ is the truth order, that allows to compare values regarding their truth. Please note that t and f are not comparable by $\leq_k$ and vice versa $\top$ and $\bot$ by $\leq_t$. Figure 6.3 illustrates the relationship of the four values and their partial ordering.

The main idea behind that approach was to explicitly represent the possible contradictions from uncertain context information during the reasoning process. Using 4V the CE can evaluate the event instances also with respect to possibly con-

tradicting information and use this contradiction to reason on the condition result.

N.D. Belnap Jr. [BJ77] has also defined logical operators to reason on the results of a statement in 4V including conjunction $\wedge$, disjunction $\vee$ and a complement $\sim$. The complement negates the truthfulness as well as the knowledge of a statement. But as a true negation operator should not create new knowledge, there was an additional operator defined for 4V to implement a negation $\neg$. This negation only affects the truth value but not the knowledge value.

In order to integrate 4V in the CE we have to do three things. First, we must map the information from the event instance and also ⋆Con to a form that can be represented as value in 4V. Second, we have to exchange the PT operators in the CE with its 4V counterparts. Finally we have to decide how to handle contradiction and ignorance results so that the navigator can make an appropriate navigation decision.

To map the information to SL we use pair of "queries" to the event instance which is comparable to the analogy first used for describing 4V. We again construct the binominal opinion for each literal in a condition ω_u^{*Con}, where u represents a desired event in the flow for that literal. The 4V CE evaluates the opinion directly against the navigation threshold $\omega_u^{*Con} \geq t_n$. Please note that the inclusion of ⋆Con is again adapted by the use of SL with the subjective opinion. This way we do not loose this elegant mechanism. The result of this first "query" is the basic information we use to construct a suitable representation of the context event in 4V. The first column in Table 6.1 lists the possible results. But this result on its own leaves no space for contradiction or ignorance. Therefore, we have to apply a second query that can make the result more plausible. Therefore, we ask if in the original unmodified event instance the event type u also was the most significant

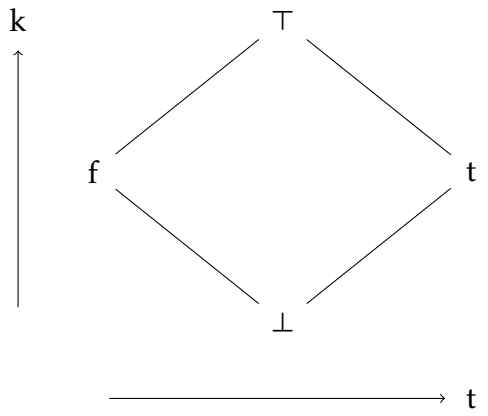

Figure 6.3.: Bilattice of four-valued logic

one $u = \hat{u}$ where $\hat{u} = \arg\max_{u \in E} I_E^e(u)$. If $u \neq \hat{u}$, i. e. it was not the most significant one, then we would have rejected the condition without the use of ⋆Con. For example consider the event type set E_α (cf. Section 3.2.1) that defines the mode of locomotion of a person. Let $I_{E_\alpha}^e$ be the interpretation for the current event instance of this event type set. $u = \textit{person running}$ is the event type in question and its probability is $0.42 = I_{E_\alpha}^e(u)$. If this is the highest probability in the event instance $\hat{u} = u$, then the result is true. If there is another event type e. g. $u' = \textit{person walking}$ with $0.46 = I_{E_\alpha}^e(u)$, then $\hat{u} \neq u$ and the result is false.

We interpret this as a suitable contradiction in order to map the context event to a statement in 4V. The second column in Table 6.1 shows the possible results of this second "query". Based on this two query results we define the mapping of the event instance to a 4V value as depicted in Table 6.1 column three.

Table 6.1.: Mapping and Interpretation of 4V values for condition evaluation

$\omega_u^{*Con} \geq t_n$	$u = \hat{u}$	value of 4V	result of the transition
true	true	t true	true
true	false	$\top$ contradiction	true
false	true	$\bot$ ignorance	true
false	false	f false	false

Having the values of 4V for the literal values of the conditions we can now continie with the second step. In order to use 4V to reason on the values we have to replace the logical operators with their appropriate counterparts from 4V as well.

Finally, we have to decide how to handle the final result of this logic, when actually making the navigation decision for the condition. Therefore, we have run a few experiments with different interpretations. The one that performed best was the relaxed execution semantics where the flow navigator could successfully navigate transitions even when the associated condition had final results of $\top$ or $\bot$. This interpretation is also shown in Table 6.1 in column four. This decision is reasonable because the uncertainty of CMS is still the major factor that can break the successful execution of the flow. Allowing also contradicting or missing information states to contribute to the flow execution, helps to reduce this influence.

We will show this in detail in the evaluation in Chapter 9. Especially the third case $\bot$ (ignorance), where we have a lack of information or ⋆Con is not helpful (e. g. for a system still in the training phase), the overall system can benefit from this interpretation. On the downside we again have to discuss the possible influence of this interpretation on producing successful false positive flow executions. The overall quality of the decisions made with 4V do not suffer and a possibly wrong decision is detected in most of the cases ($\geq$ 96%) during the next activity. Furthermore,

when combined with FEvA this kind of errors could be detected and resolved in a graceful and transparent fashion.

7. Adaptive Fuzzy Event Assignment

7.1. Preliminaries

While the mechanisms in Chapter 6 focused on the uncertainty of a single event, we will now consider sequences of events. Context information perceived by an application always has a temporal dimension: A stream of events is detected and propagated by a CMS over time. Since the class of applications we regard here is tightly coupled to a real-world process that produces events in real-time, there is a need to detect these events as they occur and assign them to the right activity. As the focus in this thesis it on the uncertainty of the event we do not cover failures caused by node failures, link or network failures and communication errors. Those errors can be resolved by standard mechanisms. When a flow executes an activity a it greedily subscribes all its event type sets ϵ_a at the CMS. While the activity is being executed appropriate context events are submitted to the flow engine with exactly one copy and stored in the event cache of the flow engine (cf. Figure 4.1). But the assignment of the event to the right activity is not always straight forward. Consider that two activities have subscribed for the same event type set but are interested in a different situation. While the event instance for one activity fits to its execution, the other may greedily consume the "wrong" context event as it has no means to distinguish if its instance was wrong or just has been classified with a low probability by the CMS due to the uncertainty in the recognition. Further the CMS itself is prone to not detecting an context event at all (e. g. due to noise or failure of components in the CMS) or even false positives that did not happen. Thus, the flow system receives an event sequence that does not necessarily match the sequence expected by the receiving flow based on the partial ordering of its activities.

In this chapter, we present Fuzzy Event Assignment (FEvA) as a means for robustly interpreting incoming sequences of context events and assigning them to the correct flow activities. FEvA deals with errors in the event sequence that lead to wrong decisions and missing information during the flow execution. The main idea is to assign the events in fuzzy and delayed fashion that reflects the uncertain nature of the received context events. First we will briefly introduce related work with respect to workflows and decisions under uncertain information. Then we describe FEvAs goal more formally giving an overview on its working principle.

After that, the main algorithms are presented and the chapter is concluded with a discussion.

7.2. Related Work

With respect to the integration of context information in workflows we already mentioned the work of Wieland et al. (cf. Section 5.1.2). The authors later extended their work to handle the integration of uncertain context in workflows with a special focus on the different levels of abstraction that context information is provided. However, they do not actually present a concrete algorithm for matchmaking between uncertain context information and the workflow activities, as we do with FEvA[1]. In fact we found no matchmaking algorithm in literature that shows how uncertain context information can be attributed to the right activities in the context of BPM.

7.2.1. Workflows and Fuzzy Logic

Adam et al. [ATV05, ATM03] proposed to use fuzzy logic to enable so called soft decisions in workflows based on the input provided to the workflow. They propose operators for workflows that allow the transformation of input information to fuzzy values and allow for a more robust and soft decision making. The approach aims to make business decisions not on arbitrarily chosen values. For example consider a process that gets the the budget and headcount as input and has to perform the necessary actions to setup the project organization as matrix or hierarchical. Basically there is no simple mapping for the combination of budget and headcount required to go for a matrix or hierarchy but it can be only formulated as crisp decision, e. g. budget $\geq 20k$ and headcount ≥ 5. With the soft decision makeing these values could be used to express the transition from one project organization to the other in an elegant fashion. This approach is also targeted for better and more natural decisions from the business perspective of the workflow. This is also the intution we have used to design FEvA. However, the authors do not consider uncertainties or ambiguities in the input information or context information in particular.

7.2.2. Fuzzy Petri Nets

Another area of related work comes from the research on Petri Net (PN). There are plenty of workflow models based on PNs (e.g. [AHH94]), and also fuzzy petri net

[1] We assume a homogeneous level of abstraction in the context information

variants have been proposed [PG94] and applied to workflows [RCMR01]. Basically all elements of a PN – places, markers and transitions – can be fuzzified and integrated into a fuzzy reasoning process. A fuzzified markers in a PN is similar to the way we fuzzify our event instances. But the context events represent external input, which has not been considered in terms of fuzzy PNs. When further compared, we avoid having a fuzzy execution state, whose semantics are difficult to define with respect of the concrete real-world application scenarios the processes are designed for. While we keep a crisp execution state and clearly defined and documented execution state, we still allow soft decisions when navigating the flow as more context events become available.

7.2.3. Event Sequence Error Model and Problem Statement

Having the necessary background information we can now formally define the error model we assume. As we already explained a flow can successfully execute if the accuracy of the events in the sequence $\mathcal{S}$ is sufficient. But there is a second condition. The events also must be also be consumed by the right activities so that the flow can execute successfully. As our uncertainty model only represents the uncertainty of one event instance we also need a model for expressing errors in the event sequence $\mathcal{S}$ that can affect the mapping of context events to the activities.

As explained at the beginning of the chapter, each activity follows a greedy schema and consumes events of the correct event type set it receives. This is fine as long as there is a clear mapping of event type sets available and the flow engine does not receive any erroneous events. In order to explain each error, we use a running example based on the following sequence $\mathcal{S}_F = (e_1, e_2, e_3)$, with the real-world values of e_1 = *person sitting*, e_2 = *person reading*, e_3 = *person walking*. e_1 and e_3 belong to the "mode of locomotion" event type set E_α, while e_2 belongs to the event type set "desk activity" consisting of *person reading* and *person writing*.

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 $\mathcal{S}$. For example, an alpha of 0.25 means that for four real events in the event sequence there is one additional false positive event. $|\mathcal{S}_{0,1,0}| * (1 + \alpha) = |\mathcal{S}_{\alpha,1,0}|$, where $|\mathcal{S}|$ denotes the number of events in the event sequence. We assume that added event instance are uniformly distributed over the sequence. The event type set of the false positive is randomly picked from the other valid event type sets $\mathcal{E}$ used in the flow, so that it can be received similar to real event instances, given the right subscription. The probability distribution of a false positive is similar to that of the other event instances in $\mathcal{S}$, i. e. they cannot be distinguished from correct events in $\mathcal{S}$ when inspecting the distribution. When adding a false positive to $\mathcal{S}_F$ the resulting sequence can

look like $\mathcal{S}_{F_\alpha} = (e_1, e_2, e_F, e_3)$ with $\alpha = 0.33$, where the false positive e_F indicates with medium significance that the person is running.

The second error type are *out-of-order events*. These become relevant, e. g. when variation in sensor data acquisition triggers the recognition of a certain context earlier at a different point in time than originally modeled within the flow. For example in the sequence $\mathcal{S}_F$ it is denoted that the person is sitting and then starts writing. However if the recognition of $e1$ = sitting gets delayed (user started writing halfway on the move), then the event instance e_2 may arrive before e_1. 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 $\mathcal{N}(0, \sigma)$. We use a normal distribution here because we assume that most events will only be delayed for a small period in time and only very few events will suffer from a large shift. For example, given $\gamma = 0.60$, 60% of the events have not been affected, but the remaining 40% are shifted in time (either way) according a normal distribution $\mathcal{N}(0, \sigma)$. The σ is chosen such that $1 - \gamma = 40\%$ of the samples are larger than ± 1. The integer part of the sample indicates the shift and direction in the event sequence relative to its original position. Given these numbers the sequence $\mathcal{S}_{F_gamma}$ can be altered like this (e_2, e_1, e_3).

Finally, there are *missed events* or false negatives that happened in the real world but have not been recognized by the CMS. The most likely cause for this is an error in the CMS or a temporary sensor failure. Let δ denote the fraction of events in a valid event sequence that have been missed. So a value of $\delta = 0.1$ results in $|\mathcal{S}_\delta| = |\mathcal{S}| * (1 - \delta)$. In the example $\mathcal{S}_{F_\delta}$ with $\delta = 0.33$ the sequence may become (e_2, e_3).

A single event sequence can be subject to all three error types and we write $\mathcal{S}_{\alpha,\gamma,\delta}$ to express its error properties. Each error type is applied and only considers the events in the original sequence. So first the false positives are added for $\alpha > 0.0$, then the original events are shifted if $\gamma < 1.0$, and finally for $0.0 < \delta < 1.0$ some of the original context events are removed.

7.3. Algorithm Overview

The main idea behind FEvA is to exploit the structural and contextual relation of flow activities and the current execution state of flows as additional information for dealing with errors in the event sequence as defined above. The errors in an event sequence $\mathcal{S}$ are critical as they can quickly drive the flow into failure and require user attention, because some event instance has been mapped to the wrong activity. However, the fact that a flow is a temporal model that also describes the possible sequences of events helps us in dealing with this problem. The goal of FEvA is to interpret an incoming, erroneous event sequence $\mathcal{S}_{\alpha',\gamma',\delta'}$ 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. During the execution of a single activity, it will eventually have candidates for all the event type sets it is 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.

Hence FEvA consists of two algorithms, one for event candidate selection and one for event assignment. Both algorithms are plugged into the event cache of the flow engine. The combination of FEvA and the event cache is referred to as *event container* (cf. Figure 7.1).

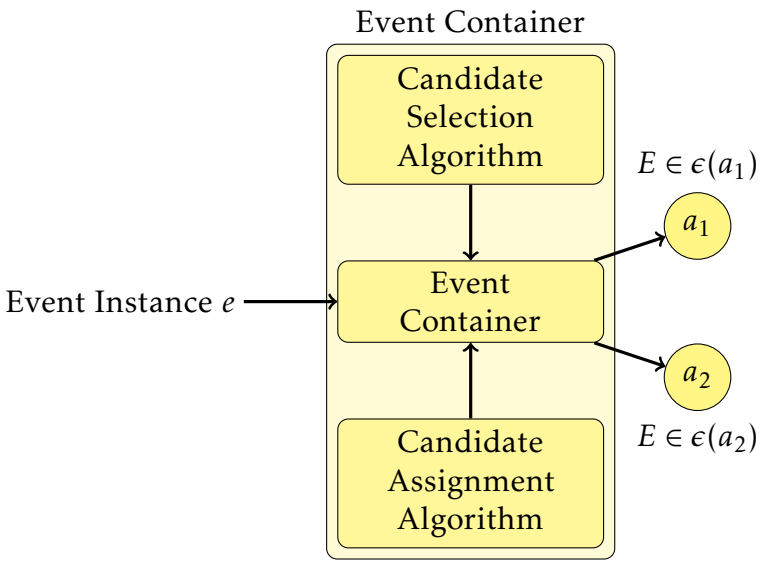

Figure 7.1.: Event Container

7.4. Flow Activity State Space Extension

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 (cf. Section 5.2). The state machine for an activity is depicted in Figure 7.2. While the continuous lines represent the original execution states and transitions, the items with dashed lines denote the states and transitions added along with FEvA.

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. But while the state space has been extended, there is no change made to the execution behavior of the flow model (cf. Section 5.2).

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 registered at the CMS ahead of time. Therefore, the risk of missing an out-of-order event that arrives earlier is reduced, because the activities also register early for their event type sets. This is necessary as the flow engine only registers event type sets at the CMS when the execution of an activity has already been started. If an event is recognized and the flow engine has not yet registered the event type sets at the CMS, it will not be forwarded to

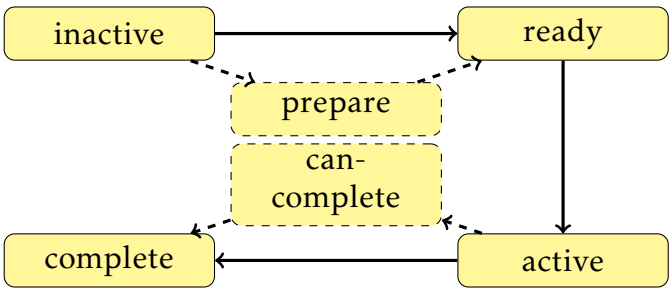

Figure 7.2.: Activity State Machine

the flow engine. Due to this change, events can now be caught by the flow engine until the prepared activity actually becomes *ready*.

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 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 7.6). To illustrate the extensions to the state space better, Figure 7.3 depicts the structure of the blood sample flow including the activity states and the appropriate sets defined by FEvA. Here the activities a_1, a_2 are already *complete*, the activities a_3 and a_4 are *active*, while a_5 is *prepared*. The activities a_6 to a_{10} are still *inactive* and waiting for execution.

7.5. Candidate Selection Algorithm

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 *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.

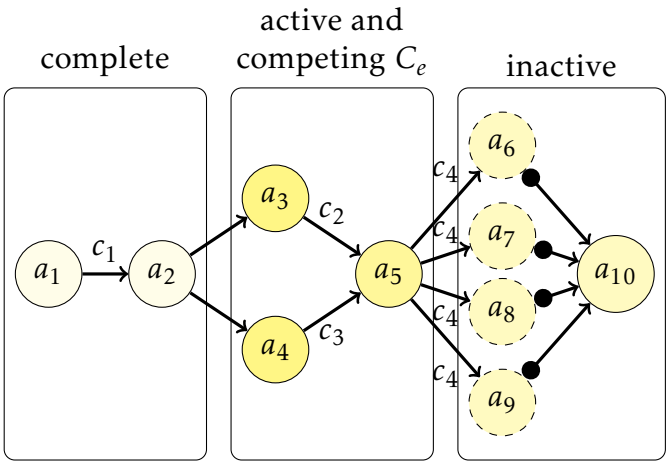

Figure 7.3.: Blood sample flow execution states according to FEvA

The candidate selection algorithm, depicted in algorithm 7.1, computes which event instances are added to the list of candidates of an activity. It notifies the activities of incoming event instances of the correct type and receives a fuzzy weighting of the event instance in return.

Algorithm 7.1 Candidate Selection Algorithm

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

Since we only get the probability distribution I_E^e for every event e, there is no hard criterion for deciding which activity e can be mapped to, but only probabilities. Therefore, we use fuzzy set theory (cf. [Zad65]) 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 containing a number of membership classes, 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 membership classes "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 membership class. We use the same membership functions μ_x based on the standard triangular fuzzy functions [Ped94] for all combinations of activities and event type sets. For example, $u \in E$ is the event type representing that the caretaker 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:

Definition 7.5.1 (Fuzzy Event Type Weighting Function) *A concise version, which includes all the membership function results, is the fuzzy event type weighting function* $\lambda : [0,1] \rightarrow [0,1]^5$. *Given* $u \in E$, $\lambda(I_E^e(u))$ *yields the mapping of the individual probability of the event type to the fuzzified membership in all five fuzzy sets, i. e.* $(\mu_{VL}(I_E^e(u)), \mu_L(I_E^e(u)), \mu_M(I_E^e(u)), \mu_H(I_E^e(u)), \mu_{VH}(I_E^e(u)))$.

For λ 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 $p > 0.55$, $\mu_{VH} = 1.0$. These values have been chosen empirically to match the overall performance of the CMS (cf. Section 8.3.2).

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 membership class 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.

7.6. Candidate Assignment Algorithm and Conflict Resolution

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 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. $(candidates(a_o, E) \backslash e) \neq \emptyset$, 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 has 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. In more formal terms $E' \in (\epsilon(a) \setminus E) : \exists u \in E', \exists e' \in candidates(a, E') : \mu_{VH}(I_E^{e'}(u)) \geq 0.0$. Given this e' is available, the equation is fulfilled and the threshold for the missing event type set E is reduced to "medium". Then other events in $candidates(a, E)$, 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. This way a major quorum of real context events have been consumed by the activity before FEvA decides to declare the other events as missing. 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 5.2.

7.7. Discussion and Conclusions

With FEvA, we have presented a new system for robustly providing context events to a workflow-based system. FEvA exploits the flow knowledge encoded in a flow

in order to assign noisy and erroneous incoming context events to the correct activities in the workflow. We have explained the mechanisms of FEvA under the given error model. As we will show in Section 9.4 FEvA 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.

FEvA is a major step towards robust workflow-based pervasive systems as it allows workflow to be executed with minimal human intervention. This is particularly relevant in application domains like health-care where workflow can support the personnel and make its work more efficient.

FEvA would be a very useful supplement for systems and environments where a lot of context information drives structured applications, such as the health-care documentation scenario we already mentioned (cf. Section 6.4 and later Chapter 8). In this scenario, FEvA significantly improves the perceived dependability of the application, advancing their user acceptance. In summary, FEvA represents a unique approach bridging the gap between activity recognition and context aware applications, dealing with ambiguities when consuming the recognized events.

Part III.

Evaluations

8. Methodology

In order to assess the effectiveness of FlowPals mechanisms, we used two different approaches. The first approach is based on a real-world case study that we conducted in a geriatric nursing hospital. This *health-care scenario* provides us a number of conditions that are relevant for the assessment of FlowPal. It includes real-world activities, a structured work environment from which we can derive a flow and rich human interaction. Furthermore we can apply real context recognition technology to recognize the activities with all involved accuracy issues. Hence, this scenario provides us with all the challenges a FlowPal wants to tackle. On the downside we can only perform a rather limited number of experiments in the real world and some of the experimental conditions remain fixed. This is especially unfavorable as we only observe one distinct flow-based application and hence only a single flow structure. As FlowPal aims for improvements independent of a certain flow structure this would introduce a major bias for the evaluation results.

To overcome this limitation, we tried to come up with a large number of flows in our second approach, that have the same basic properties as the one observed in the health-care scenario but a different structure. Therefore, we used so called *workflow activity patterns* to generate flows that in general have the same properties and can be compared to the real flow from the health-care scenario. The workflow activity patterns (or simply patterns) have been identified in a number of workflows from different application domains and can be divided in human-intensive and system-intensive patterns, where we focused on the human-intensive ones.

The two stage approach is beneficial, as we can perform evaluations in both settings. On the one hand, we have a very specific and realistic scenario with real-world context data. On the other hand, we can execute a large number of synthetic experiments using the generated flows. Therefore, we can thoroughly analyze the performance of Flow Pals mechanisms.

In the remainder of this chapter we first present a detailed description of the health-care scenario in Section 8.1, including scenario setup, recorded sensor data, and modeling the respective flow based application. In the following Section 8.2 we introduce the workflow patterns and our method to generate a large number of synthetic test flows. After that we compare both evaluation strategies and discuss

their individual contribution to our evaluation results. Finally, we describe our simulation setup and the relevant metrics in Section 8.3.

8.1. Health-Care Scenario Case Study

The health-care case study was conducted in the medical hospital in Mainkofen, Germany. We visited a closed ward for geriatric nursing. The ward is an intensive care station for elderly people suffering from dementia and similar old-age diseases. Each of the patients needs care around-the-clock. The treatment usually includes an individual therapy, drug treatment and possibly assistance for basic daily routine tasks such as eating, dressing and personal hygiene. There are usually up to two physicians and up to six caretakers at duty. The ward has a capacity of up to 20 patients.

First of all we studied what real-world processes are executed in the hospital and which one process is suitable for the case study. The process we wanted to observe should met the following three requirements. It should have a basically fixed structure so that it can be easily represented as a flow-based application. Furthermore the process should exhibit a relatively high repetition rate so that we can observe a sufficiently large number of instances executed during the case study. Finally, the process execution should include rich human interaction, that is necessary for its completion. On the one hand this interaction is challenging for activity recognition, but on the other hand the process can be documented directly and time-saving in comparison to the current way, where the caretaker is manually updating the patient records at the end of the shift.

Treatment plans have been a good candidate process but as they cover time spans of weeks to months it has been infeasible to study them in the health-care scenario. So due to the time constraints of the study we chose to investigate the daily morning-routine of the patients. This way we could guarantee to observe a sufficient number of process executions yielding statistically relevant results for the observation. Furthermore considering the range of tasks for each patient, the process structure is quite stable and has only slight variations, when compared to the individual treatment plans. The execution of the morning routine involves intensive interaction between the patient and the caretaker. Therefore, the morning routine process satisfies all the aforementioned requirements. However, it was not available as a modeled workflow but only as the sum of all guidelines, working routine descriptions and experiences of the caretakers.

A single instance of the morning routine flow covers the wake-up of the patient, helping the patient getting washed, dressed, having breakfast and the administration of the individual medication. One caretaker is usually responsible for three

to five patients per shift. As the caretaker is involved in each of these patient processes and the caretaker is the more active part during process execution, we chose to observe the caretakers and recognize their activities when interacting with the patients. This way the recognized activities are ordered with respect to the – possibly intertwined – execution of the morning-routine processes of the individual patients.

The process execution has to adhere to the medical guidelines in force. All activities performed (e.g. treatment, medication) stringently have to follow the guidelines and the hygiene standards must be kept. Furthermore, the results of some activities, like examinations and medication must be documented. As we were at the hospital these guidelines could only be kept on a best effort basis and the documentation was patchy, manually and potentially error-prone.

Applying a system using adaptable pervasive flows in this institution pursues two purposes: First, the activities performed during the morning routine process 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.

To demonstrate that these goals are achievable with today's context recognition technology and the methods provided by FlowPal we performed the following steps, which we describe in the next two sections. First, we defined how to record context data observed on site during the study (c.f. Section 8.1.1). Then, we processed the collected data off-line to determine the overall quality of activity recognition that is possible with our chosen recognition approach. This is not an original contribution of this thesis but work from our colleagues Gernot Bahle, Agnes Grünerbl and Kai Kunze. Thus, we will only give a brief summary here and refer to the original work [KBNL11] for the full details. The results have directly informed our evaluations and we will discuss this in Chapter 9. Finally, we had to to derive an executable process model. There are two options to do this. Either a domain expert creates the process model or a technique called *workflow mining* is applied to recorded data to extract the workflow from the observations made. As we are no health-care domain experts we opted for the second solution. The results of the flow mining are presented and discussed in Section 8.1.2.

8.1.1. Study Setup

The traces we obtained from the nursing ward represent the daily morning routine of individual caretakers. Each caretaker was supplied with a commodity smartphone that we use as sensor platform for the data collection. The smartphones were carried in the coat pocket. The sensor readings available in the resulting

traces are (1) received WiFi signal strength, (2) magnetic field strength, (3) acceleration, (4) inertial movement and (5) sound. The WiFi readings, inertial movement and magnetic field information were used to estimate the indoor position of the caretakers on a room-level granularity and facing direction. We further added location hotspots to the rooms where applicable, e.g. each patient room had an extra location hotspot for the en-suite bathroom and the breakfast area had a location hotspot for each table. Overall we had 33 locations to distinguish. Some of the locations were further grouped together to an area, where each area represents the patient rooms and including hotspots that a caretaker on duty is responsible for. The layout information on the ward are depicted in Figure 8.1. The most relevant sites for the morning routine flow are the patient rooms $Z_1,\ldots,Z10$, for washing and dressing, the breakfast room SR and the recreation room A for waiting periods. The ward office PS is also important for the official documentation tasks. The area 1 and area 4 are also highlighted in the floor plan. Please note that the necessary WiFi infrastructure was already present in the hospital, so there was no need to deploy further infrastructure.

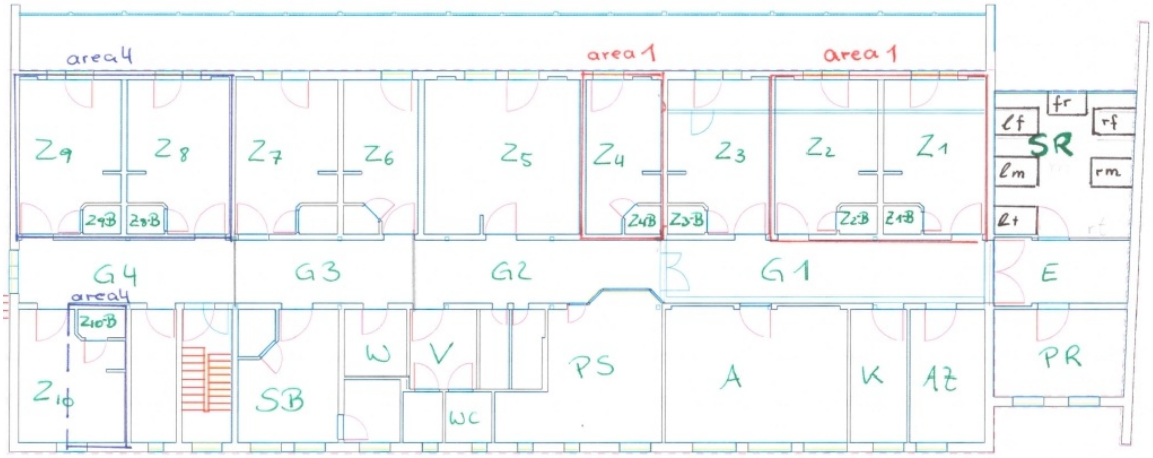

Figure 8.1.: Floorplan of the Mainkofen Ward

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 these modalities were combined.

As each of the caretakers on duty carried a smartphone, we could record all available data for a given day. Further, we manually labeled two or more traces each day with annotations describing the respective real world activities and also flow activities. Therefore we had two distinct sets of traces. The unlabeld traces were used as a test set for the activity recognition. The labeled traces were used for two purposes: categorization of context information for the recognition and tagging flow activities (cf. Section 8.1.2). A training set was created from the context recogni-

tion labels, that allows us to classify the sensor data. This set includes labels that specify the beginning and ending of sounds in the environment (e. g. "shower", "electric shaver"). Further the location of the nurse based on symbolic hot spots, as well as her basic movements (e. g. "pick up something", "shave the patient") are noted. Finally, we kept track of the nurse's resource usage (e. g. "towel", "stethoscope").

The training set has been used to train the classifiers for indoor positioning and activity recognition on the traces. For the indoor positioning we achieved a correct recognition rate of 75,19%, e. g. in three of four cases the classifier is able to discern the location of the caretaker correctly for the described hotspots. For the activity recognition we achieved only an average recognition rate of 48%, so that the classifier was able to discern the correct activity in one of two cases. However, the activity recognition rate varies greatly depending on the specific activity. In particular, activities that can be associated with a specific background sound (e. g.. "shower the patient") could be recognized far more accurately (with recognition rates up to 83%). However activities lacking this sound signature have been be recognized with much lower accuracy ranging from 31% to 56%. While the activity recognition methods and techniques are no contribution in the scope of this thesis we refrain from presenting more detail on the methods here but refer to the original work of our colleagues and the respective publication. [KBNL11][1] However, the overall data collection (setup, preparation, tool provisioning) and especially the post processing described in the next section have been designed and conducted by the author of the thesis.

The results on the context recognition rates from this part of the study provide us valid assumptions on the data quality a robust flow system has to deal with in order to execute the workflows correctly. Therefore, we used the values to inform our evaluations with realistic input information (cf. Section 8.3 on page 148).

8.1.2. Flow Mining

As we started with our study, there was no explicit step-by-step process description (e.g. executable workflow model) available for the hospital personal. They have guidelines which they interpret according to their skills, knowledge and experience. Further, each caretaker adheres to the medical regulations and hygiene guidelines but each one also exhibits some individual daily routine. This leads to the following implications. First, there exists a lot of structural knowledge in the scenario that can be used to model an explicit flow definition. Second, we have

[1] This work has been accomplished in collaboration with researchers from the Embedded Systems Lab, University of Passau.

to extract and define the flow because there exists no written process definition yet.

In order to come up with an executable process model we need a formal representation of the performed workflow. Usually a domain expert is responsible for creating this formal representation in a given workflow modeling language but there was no suitable expert available for our study. Therefore, we used tools for process mining to derive an executable workflow model for our evaluations[2]. In general, a flow mining algorithm performs an analysis on certain *event logs* that record the execution of activities that should be covered by the created flow model. The basis to create these event logs were the flow activity tags from the labeled traces we recorded during the study.

The labels for flow activities are fixed and have been predefined during the preparations of the case study with on site testing. In each trace we labeled a caretaker 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: (1) The 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. (2) During the morning hygiene, the caretaker helps the patient with getting up, washing and dressing. (3) Following that the caretaker helps the patients having their breakfast. (4) Finally, the caretaker supervises and assists the patient taking his daily morning medication according to the patients capabilities.

The caretakers perform the first two steps of this patient flow usually in a sequential manner for every patient. This changes during the breakfast where a lot of interleaved patient handling can be observed. This greatly depends on each of the patients autonomy and thus the degree of support a caretaker has to provide. Due to this behavior we assigned the flow activity labels to the respective patient, which results in a sequence of activities that have been performed on a per patient granularity. These sequences of activities have been used as so-called *event logs* as input for the flow mining algorithms. In total, we labeled 32 datasets from 15 different caretakers, where each dataset covers the care of 3 to 5 patients, yielding a total of 135 observed patient flow executions.

In order to extract the workflows from the event-logs we applied two different flow mining algorithms that are provided with state of the art process mining software [DMV+05].

The first algorithm is based on PNs and is suited for highly structured processes that exhibit low flexibility. It tries to generate a PN from the event log that can capture all the sequences, which occur in the recorded event trace. PNs are a

[2]This work has been accomplished in collaboration with researchers from the Institute of Architecture of Application Systems, University of Stuttgart, who supported us with details on generic workflow miners and tools

well standardized and formalized representation for workflows (cf. Section refsubsec:flowModeling). However, the mined flow model from the full data set results in a very large and complex net. This net essentially captures every execution path performed by one of the caretakers during the study. A visualization of full PN did not provide us with any insights on the overall flow structure and the necessary flexibility. This shows that the process we want to model indeed exhibits a very high variability for the following reasons. First, each caretaker has its own individual routine. When these routines are mixed together – this is what this mining algorithm tries to capture – the result can be expected to be very complex. Second, for some caretakers the manual labeling could be accomplished more accurately than for others, which of course leads to issues with data quality. When the mining was performed on a subset of the traces for a single caretaker in a single area only, the results became somewhat better, especially for the hygiene and examination tasks. However we had only a very small overlap of caretakers and regions. There were at most two traces (e.g. 10 patient flow executions) where the same caretaker was on duty in the same area.

As this first flow mining approach could not come up with a reasonable result flow we focused the mining activities on a highly structured and standardized part of the morning routine. Therefore, we reduced the number of tasks covered in the event logs and focused our efforts on eight different activities from the morning examination. These activities include measuring pulse and blood pressure of the patient and noting the results of booth examinations. Please note that we again used the event logs of the mentioned activities from all 135 recorded traces.

The resulting PN is depicted in Figure 8.2. For each activity we tagged, there are two transitions in the PN, one denoting the start, the other the end of the activity. The Graph basically shows us two things. The process starts (left-hand side of the diagram) and forks directly into one of the three main routines, either applying the cuff first or searching the pulse for the pulse measurement (feeling by hand) or searching the pulse for the BP measurement (using the stethoscope). The process ends after the "write results" activity has been completed. In-between a lot of different paths have been recorded and only for the beginning and ending of a specific activities.

This reduced set of activities still leads to a complex PN, providing us with little aid understanding the scenario. This illustrates that standard mining techniques based on classical workflow and highly structured models lead to large and basically unreadable workflow descriptions in our scenario.

The second mining algorithm we applied, is a so called *Fuzzy Miner* [Pro], that has been especially designed to mine processes with less structure. This also includes processes with unstructured or even conflicting behavior. Both can be observed in

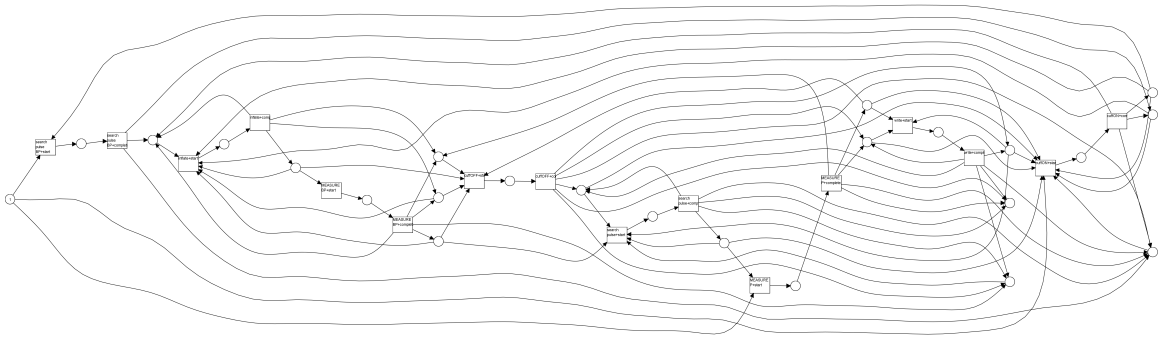

Figure 8.2.: Workflow mined as Petri Net from the traces

the traces we recorded due to variations in daily routine, interruptions and repetitions of activities. The mined flow from the fuzzy miner is depicted in Figure 8.3. It shows the activities and transitions between activities that have been observed in the traces. The thickness of the arrows represents the relative transition probability to the next activity. This provides us with a much clearer understanding of the usual execution of the examinations. For the sake of clarity, we have omitted transitions with less than 5% significance. In the figure, the basically sequential structure of the process as well as the flexible execution order can be observed more easily. We see that the process starts either in the blood-pressure or in the pulse measuring sequence. Writing down the results happens either after each examination or at the end of the process. Further, there is no noteworthy interleaving between these two examination sequences. The self transitions denote that some steps have to be repeated. This is especially obvious for the activities "apply cuff" and "search pulse for blood pressure". However, as we have removed some of the transitions this figure does not capture all the execution paths that were taken during our observations.

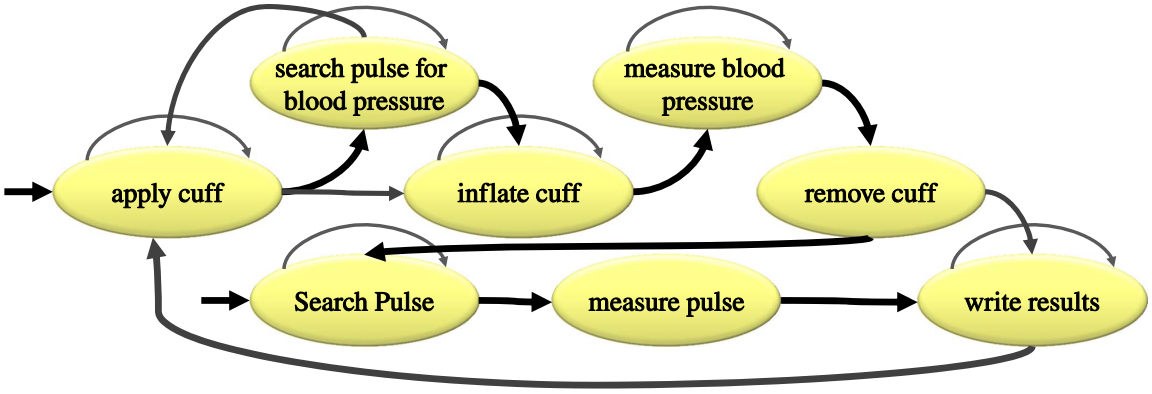

Figure 8.3.: Workflow mined as Fuzzy Transition System

Finally, adopting the gained insights on the process structure, we modeled the process ourselves using BPMN as state of the art graphical process modeling notation.

The resulting process is depicted in Figure 8.4. This model is complete with respect to the performed activities and captures the most common path through the mentioned activities. However, this flow does not allow any deviations from the modeled behavior. Therefore, it could not be used in practice in the hospital as basically every caretaker is executing the same process in his own way and with some variations each day. This clearly shows that we have to apply more flexible modeling techniques, that can handle these variations at run-time.

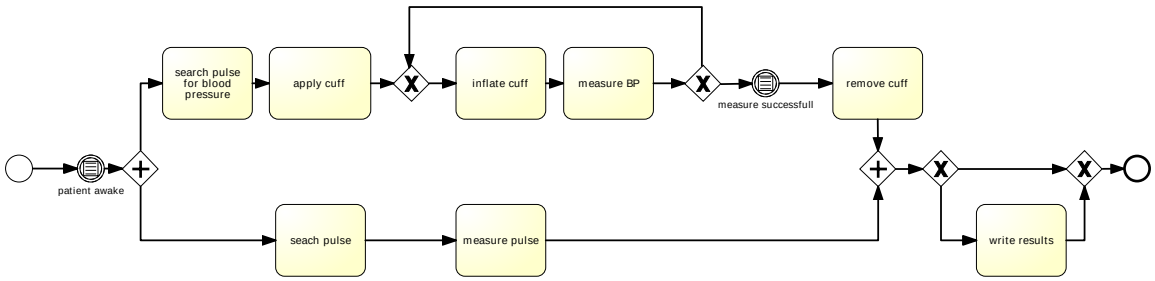

Figure 8.4.: BPMN Workflow

The HFM that we presented in Chapter 5 on page 83 provides us with a useful tool for more flexible modeling of our scenario process. The flow model of the morning examination activities is depicted in Figure 8.5. In this representation the process captures the two individual examinations and allows for the repetition of single or multiple activities in each of examination paths. Further, it allows multiple executions of the "write results" activity. On the one hand, we are able to represent most of the execution paths that have been observed. On the other hand, we still maintain some structure in our process definition. Of course, this

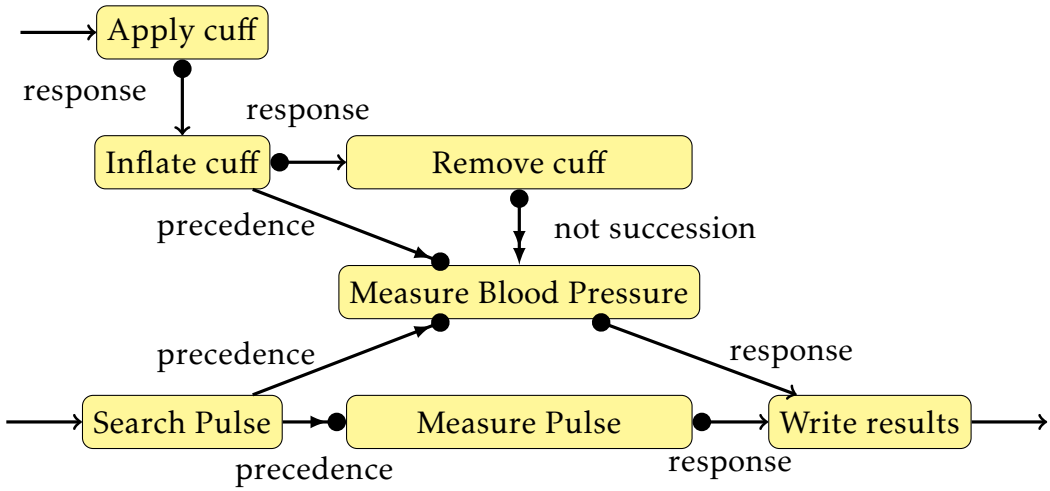

Figure 8.5.: Morning examination activities modeled with the HFM and constraints

structural information can be used to inform FlowPal and thus aid in providing more accurate context information.

Based on this final process we have a suitable real-world scenario modeled as an executable hybrid flow model that we can test the FlowPals algorithms against. However, we also have to show, that the FlowPal algorithms are independent of the structural information encoded in this specific scenario. Therefore, we will describe how we could create a large number of structurally different workflows as test stimulus for FlowPal.

8.2. Pattern-based Flow Generation

As we have seen in the previous section, we had to spend a great effort to analyze just one scenario. As a result we came up with a satisfactory flow model for the application scenario that covers all the requirements, which we defined in Section 8.1.

At the start of the chapter we motivated, that we would need a large number of structurally different flows to assess FlowPal. But a case study is not a feasible tool for creating a large number of structurally different flows. So we tried to create them in an automatic fashion.

The key for this creation have been so called *workflow activity patterns*. They have been identified as building blocks for flows in various application domains, which are also centered on human activities. First, we use these patterns to inform a *flow generator* to compose executable flow models that cover a wide range of different structures. In a second step, we generate a so called *execution model* for a single generated flow model. The execution model provides us with an arbitrary number of valid event sequences to drive the execution of the respective flow model.

In the remainder of this section we will first introduce the workflow activity patterns in more detail. After that, we describe the working principle of the flow generator and then explain the creation of the execution model in Section 8.2.1. We conclude the section with a comparison and discussion on the properties of the flows we generated with the generator and the flow (including traces) that we captured in the geriatric ward.

8.2.1. Flow Generation

We have seen in Section 2.1.2 that from a technical perspective workflow patterns can be used to describe the capability of flow modeling languages. We briefly discussed three types of patterns relevant for this thesis.

1. The control flow patterns that provide common control flow structures.
2. The service interaction patterns that cover inter-process as well as human interaction.
3. The data flow patterns that describe common data dependencies found in flows.

According to Russel et al. [RAHE05, RHEA05], the patterns are useful to measure the expressiveness and modeling capabilities of existing workflow languages and workflow management systems. So basically these patterns could be also applied to the flows and the flow model used for FlowPal.

However, for our purpose we are in need for patterns, which represent typical human behavior that is encoded in workflows. Fortunately, there has been a study that has studied exactly those kind of patterns in this research area. Thom et al. [CITR08] conducted a study where they investigated the existence of *workflow activity patterns* workflows. The authors analyzed about 200 different workflow models. The patterns they found could be classified as human-intensive and system-intensive. The human-intensive patterns cover activity sequences that involve one or more human participants and in general interaction human interaction with the workflow, while the system-intensive patterns basically represent interaction within the workflow managment system itself and between other workflows.

Overall the existence of seven different human-intensive patterns has been proven by the authors. Based on these seven patterns, all of the studied workflows could be reconstructed, i. e. for the analyzed flows the patterns are complete.

In the following we will briefly introduce each pattern and present its structure using the imperative flow model we introduced in this thesis in Chapter 5. The description of the patterns is informed by the original authors [TRI09]. For illustration purposes we highlight the main path of execution of the process with an additional dashed transition where appropriate.

These seven patterns are:

- Approval: The Approval Pattern describes a pattern of activities that cover a human decision approving some document or action in a process. For example a doctor wants to conduct a surgery. Therefore the doctor needs the consent of the patient. When the patient has approved his consent he in turn

informs the doctor who documents the success of the approval and takes further actions. The structure of the approval pattern based on the example description is depicted in Figure 8.6.

- Bi-directional Performative: The pattern covers sequences containing an activity request for a different actor that receives and full-fills the request and then answers to the request. The recipient may start additional activities (or patterns) to complete the request. In the meantime the original actor must wait for a response. As example we consider a blood sample analysis laboratory as service provider for a doctor. The doctor has a blood sample that he needs analyzed and initiates the request at the laboratory. The laboratory receives the request (and the blood sample) and performs the request (possibly asking for additional patient data). Finally, the laboratory notifies the doctor on the analysis completion and the doctor can continue with its original sequence of activities. Please note that the doctor can spawn multiple bi-directional performative actions at the same time, waiting for all of the requests to complete. The pattern is depicted in Figure 8.7.
- Decision: The decision pattern is one possible extension of the aforementioned Bi-directional performative pattern. After the original actor has received the results of the started actions, there may be a number of conditional paths to choose depending on the received results. For example after the results of the blood sample analysis have been passed to the doctor, he could conclude that the patient is either healthy (no further treatment) or that subsequent follow up checks or preventive or curative actions have to be started. The decision pattern is depicted in Figure 8.8.
- Information Request: The information request pattern is quite similar to the bi-directional performative pattern. Here the originating actor is requiring only some information from a human or software agent before he continues its own process execution. In comparison to the bi-directional performative

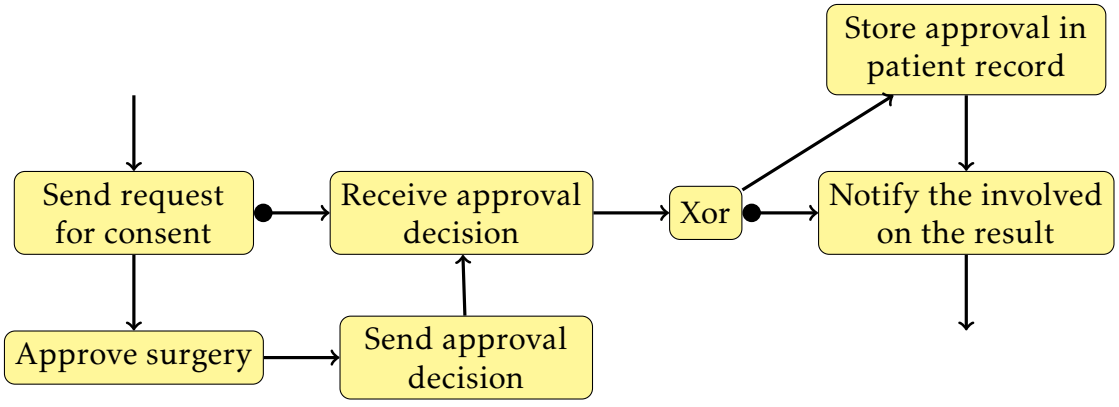

Figure 8.6.: Approval Pattern

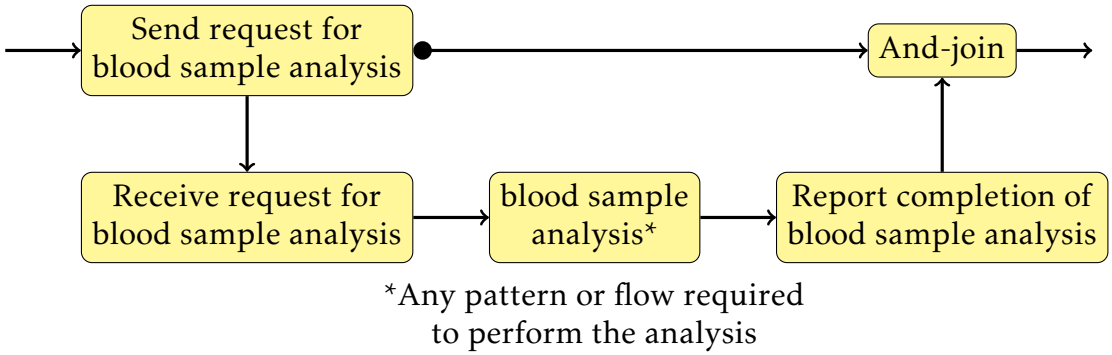

Figure 8.7.: Bi-Directional Performative Pattern

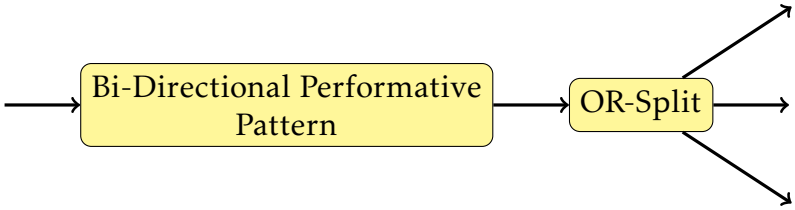

Figure 8.8.: Decision Pattern

pattern no additional activities may be performed by the recipient of the information request. This would be the case, if the doctor would be reviewing the results of a blood sample analysis conducted in the past. He issues the information request and receives the response by arbitrary means (mail, web-service, phone-call) but without the requirement to perform additional actions at the laboratory side. The information request pattern is depicted in Figure 8.9.

- Notification: The notification pattern informs one or more actors of the results of certain actions e. g. the completion of an activity or process. In our example the doctor could notify a ward and the patient to inform them both that an inpatient treatment has to be conducted. Figure 8.10 depicts the Notification pattern.

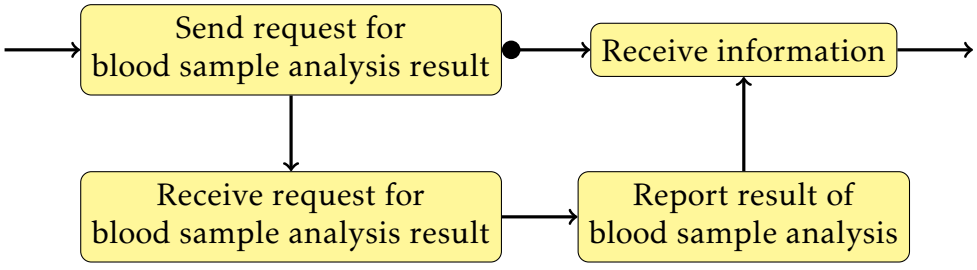

Figure 8.9.: Information Request Pattern

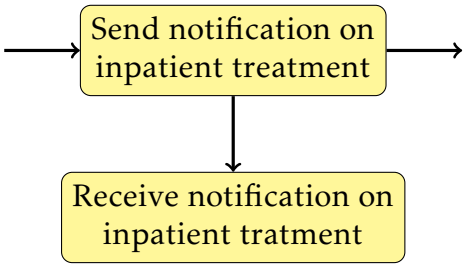

Figure 8.10.: Notification Pattern

- Question-Answer: The question-answer pattern allows the actor to query one or more recipients for additional information that the original actor requires. Before the actual questions are communicated, he first identifies the correct role of the expert to send the question to. Given the blood sample analysis again, the doctor may require information how long the analysis will take before he can send the blood sample. So, first he will identify the organizational unit that can answer the question and then acquire the information using the same pattern structure as in the information request pattern. The full pattern is depicted in Figure 8.11.
- Unidirectional Performative: This pattern covers the start of a new sequence of activities. Therefore, the original actor requests the recipient to perform this request. The doctor has completed his patient treatment process and now requires the billing department to perform the billing process for the hospital. As the doctor is not directly involved in this process or its outcome, he is free to perform other activities. the pattern is depicted in Figure 8.12.

On their own these patterns provide little use for the generation of workflows. But in following work, the authors [TRCI08, TRC+08] came up with a recommendation system for a process modeling expert. While the process modeler de-

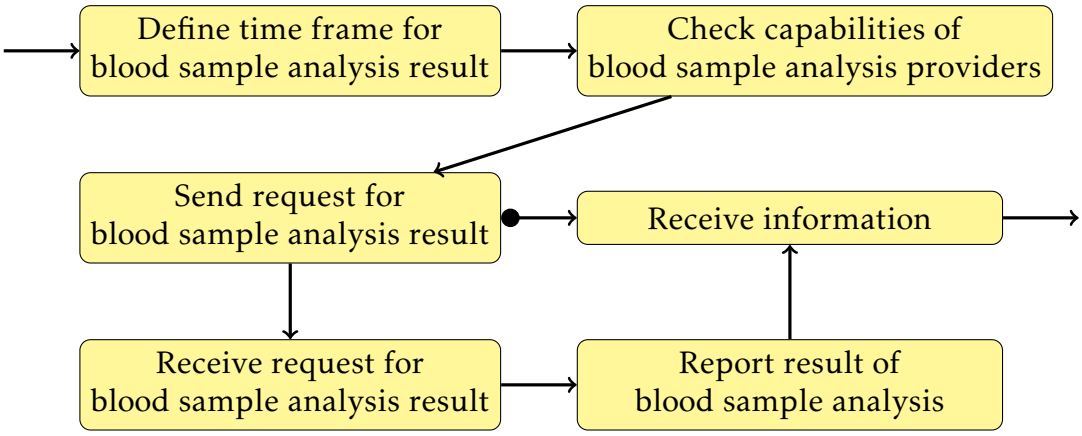

Figure 8.11.: Question-Answer Pattern

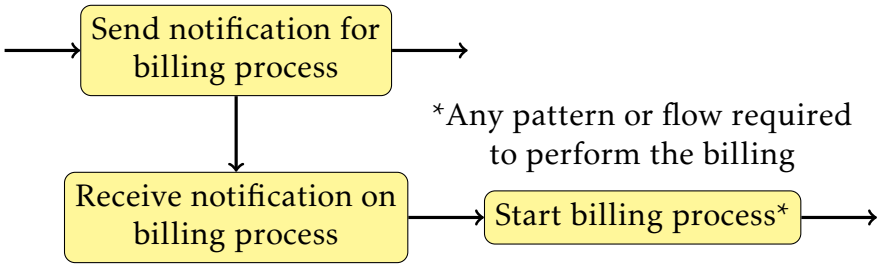

Figure 8.12.: Unidirectional Performative Pattern

signs a new process the recommendation system suggests patterns that will most likely be added next, thus simplifying the overall modeling. To build this recommendation system the authors analyzed the workflows again and extracted the co-occurrence probability of the identified patterns. For each of the seven patterns the co-occurrence probability yields the probability distribution which of the seven patterns will statistically follow. Based on these probabilities they could make educated suggestions to the process modeler. We use them in a similar way.

Having both the set of human-oriented flow patterns and their co-occurrence probabilities relative to each other, we have enough information to inform the automated flow model generator. We built a probabilistic grammar to compose workflow models using the presented activity patterns. Furthermore we added some additional control parameters. These parameters are

- the number of activities in the flow to control its overall size, given by an lower activity bound that must be reached and an upper activity bound that the generated flow should not exceed.
- the nesting probability to add additional activity patterns at the internal extension point of the bi-directional performative pattern.
- the normally distributed average degree of decision splits in decision patterns.

Based on a given configuration of these values the workflow generator works as follows. A random pattern is chosen according to the overall occurrence probability of the patterns. Most of the patterns are single-entry single-exit patterns, providing us with a single *extension point* to add further patterns. The exceptions are the unidirectional performative and the bi-directional performative patterns, which provide two extension points, thus forking the generated workflow. For the unidirectional performative pattern we allow growth on both extension points. For the bi-directional performative, we restrict the nesting of patterns within using the nesting probability. Another exception is the decision activity that explicitly introduces a split activity, where a large number of parallel execution paths may

originate. The number of paths the generator creates is determined by the normally distributed average split degree. The higher the split degree is, the larger becomes the number of parallel execution paths in the workflow, which the generator creates. Each path that does not join the back eventually with some other, has to be interpreted as one outcome of the process that is independent of the others. The number of these loose ends have to be decreased to a reasonable amount at the end of the generation process. To achieve this, the generator adds join activities. The join activities are created when the workflow in generation already contains more activities than the lower activity bound specifies, but there is still space for more activities until the higher activity bound is reached. Each of the join activities connects two of the still open extension points. When only a single extension point is left, the generation is finished.

Equipped with this basic version of the generator, we consider the validity of the generation process. The workflows are generated by activity patterns only, as they have been observed in over 200 real-world flows. The scenarios, which the patterns originate from, cover flows of companies from different application domains such as insurance, banking, medical and industry. The selection of patterns has been informed by real-world co-occurence distribution of the patterns. Each of the used patterns has been classified as "human-intensive" [CITR08] pattern, denoting that this kind of activity structure involves human interaction. These arguments provide us a good foundation.

We verified the quality of the generated flows, with respect to the activity pattern distribution and the other system parameters by creating a large number of them. We started to vary the flow size from flows with five activities (minimum to create all patterns) up to flows with 250 activities (large upper bound on observed traces in case study). In total we generated 10.000 flows for each flow size. Then, we checked three properties of the flows.

First, the generated flows must not violate the activity bounds, which can simply be achieved. Also the average flow size is closer to the upper activity bound the larger the target flow size is. As the number of total activities increases, so does the number of possible extension points that can be affected during the last phase of the generation, where then more join-activities are added.

Second, the flows must adhere to the given pattern distribution. The results are depicted in Figure 8.13. There, average and stacked percentages for each pattern type in the generated flows are shown. On the left side are the pattern frequency distributions for the flow sizes ranging from 5 to 250 activities. On the right side is the reference frequency distribution as presented by Chiao et al. [CITR08].

The graph shows that for an increasing flow size we get slightly more approval and bi-directional performative patterns compared to the reference distribution.

This overweight is compensated by a reduced number of decision patterns, which occur 5% less often than predicted by the reference distribution. This is due to the two extensions we made to control the flow generation. While appending patterns adheres the co-occurrence distribution, nesting a pattern uses the distribution for selecting the first pattern of a flow. And this distribution favors approval and bi-directional performative patterns, while decisions are less likely to occur as a first pattern. The other patterns occur within 2% of the predicted frequency. The closest match to the reference distribution can be observed at a flow size of about 35 to 40 activities. There, approval and bi-directional performative are still less than one percent of the target. Also the decision pattern contributes more than 15% to the overall patterns compared to the target of 20%. This documents that the flow generator matches the structure of the analyzed real-world flow models quite closely, but has minor deviations due to lack of data on probabilities for nested patterns.

Third, we investigated the degree of the decision splits, which the generator produces in the flows. The results are depicted in Figure 8.14. It shows the average number of parallel execution paths that originate after a decision for an increasing flow size. The benchmark degree of the decision splits is four and also depicted in the figure. The choice is arbitrary, but is somewhat close to typical variations that we observed in the healthcare scenario. The graph shows that the generated flows are slightly below the target for flow sizes of less than 50 activities. Due to the limited size, the generator has not the required degrees of freedom to fully achieve the average. The lower activity bound is usually reached before any large split can be generated. For larger flows up to 90 activities we stick very close to the target. When the flow size is increased even further, the average decreases slightly similar to the occurrence of the decision pattern in the overall pattern frequency distribution.

When we look in more detail at the split degree distribution for a specific flow size, we see the expected normal distribution, that is centered at the target split degree. The distribution for a flow size of 50 activities is depicted in Figure 8.15.

Overall, we can conclude that the workflow generated by the flow generator are structurally similar to the investigated flow of the hospital scenario and also exhibit a basic amount of choices.

In this state, the generator provides us highly structured flow models that consist only of strictly connected activities. In order to have also workflows that are coupled more loosely and cover at least a part of the design space of the hybrid flow model, we extended the generator. We added an additional creation step that allows us to add so-called *constraint based activities* that are only tied to the generated flow by constraints. As we will see in the next section, the creation of an execution model for the hybrid flows is computationally very expensive. This is due to the

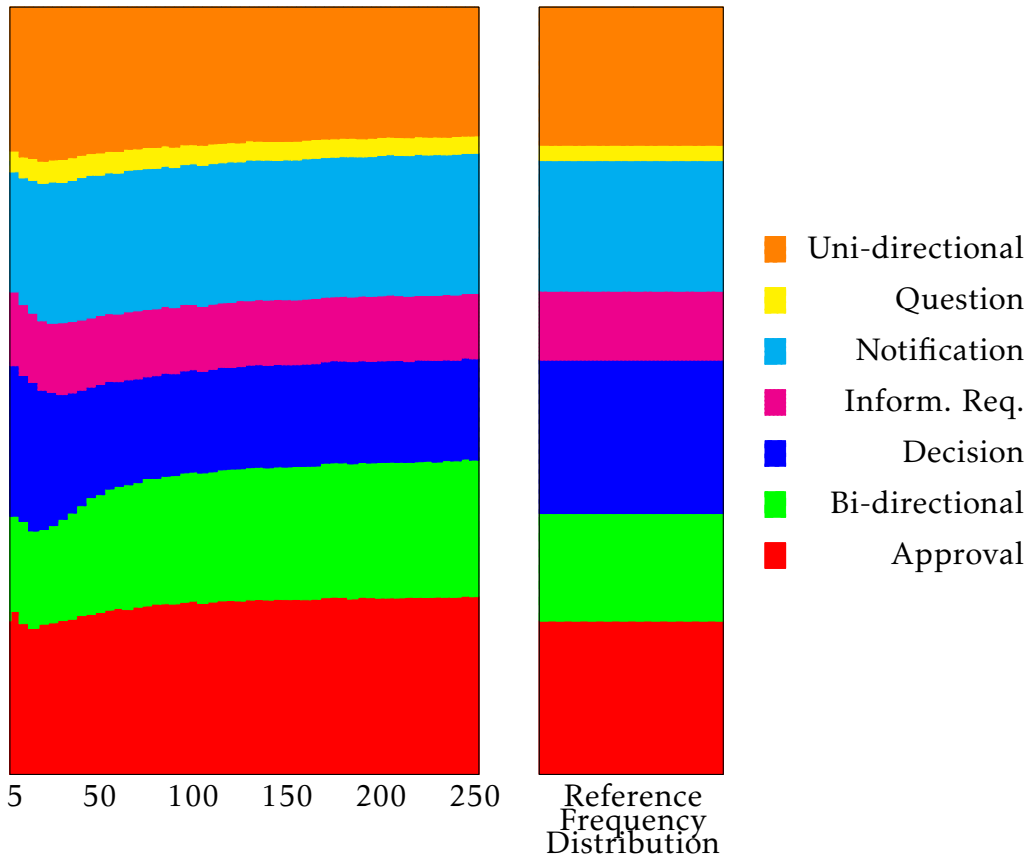

Figure 8.13.: Flow generation: pattern distribution verification

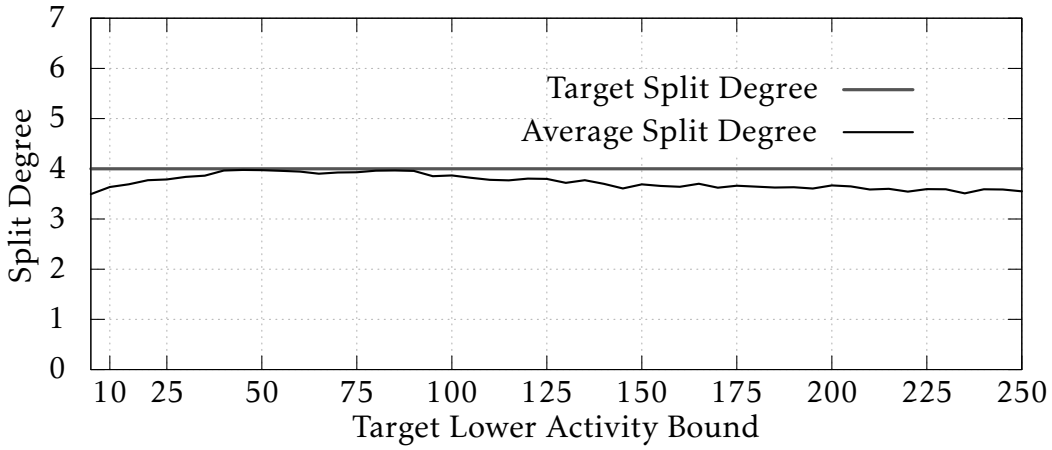

Figure 8.14.: Flow generation: average generated vs. target split degree

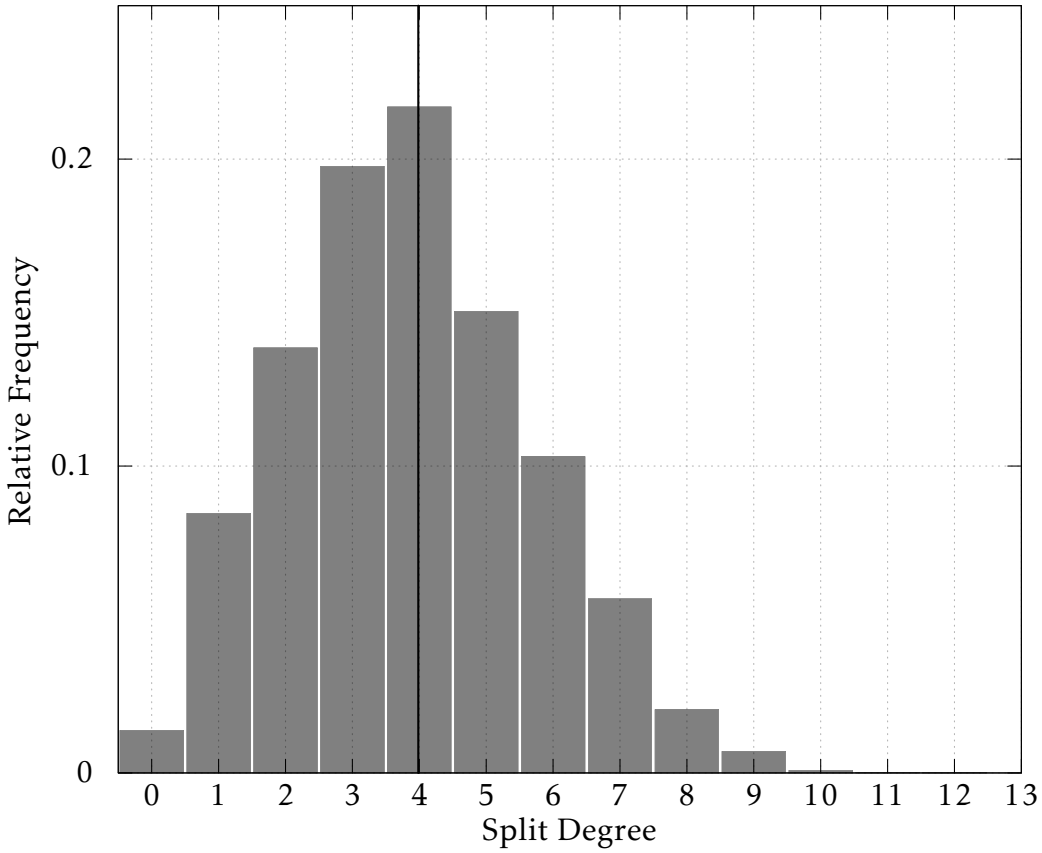

Figure 8.15.: Flow generation: split distribution verification

the automatic verification of randomly created constraints, which dictates a feasible limit of just a few constraints. Because we wanted overlapping constraints for those additional activities, their number had to be even smaller. In order to keep the ratio of constraint-based activities reasonable to the rest of the activities, we also limited the overall flow size.

The range of values we used for the generation of the workflows is discussed in Section 8.3.2 where we also introduce all other system parameters of the evaluation.

8.2.2. Event Sequence Generation

Having flow models for our experiments, we are still in need to drive individual executions of these flow models. Therefore, a number of event types and respective conditions must be defined for the generated flow models and an event sequence S must be simulated that fits to the event types and conditions. To achieve this we build a so-called *execution model* $\mathcal{EM}$ that can provide an arbitrary number of valid

sequences S for its respective flow model. Note that the event types and conditions are fixed for an single execution model.

To build an execution model $\mathcal{EM}$, we use a generated flow structure and a *split distribution* P_{split} for the decision patterns. The execution model then can generate event sequences S, which represent a successful execution of the given flow model, choosing some random execution path.

But not only the execution model must be capable to execute the flow, but also emulate some kind of "human habit". The split distribution introduces some bias when executing a decision pattern using the same execution model. Thus, some execution paths and combinations are executed more regularly. The same split distribution is also used to favor the approval or reject path in the approval pattern, but the effect on the overall flow execution is limited there, because the approval pattern is single-entry single-exit.

The mechanism to control the "human habit" in a given execution model works as follows. We use the split distribution to determine for each decision the taken execution paths based on an individual probability window.

More formally, let ψ denote the number of outgoing execution paths after a decision pattern. Then the split distribution function P_{spilt} yields an ordered ψ-tuple with probability windows for each execution path.

$$P_{split} := d_{decison}() \rightarrow ((low, high)_1, \ldots, (low, high)_) \tag{8.1}$$

Furthermore, let $P_{split}(\psi)_i$ denote the i-th pair of the associated ψ-tuple. When the execution model decides which events need to be generated for a decision pattern, it draws a random number R in range $R = [0.0, 1.0]$. Then, the execution model generates events for the i-th execution path iff $P_{split}(\psi)_i.low < R < P_{split}(\psi)_i.high$.

The execution model does not know the details of the activity patterns following the actual decision. Thus, there is no real semantic involved in this biased decision path selection. Therefore, the behavior the execution model creates does not match to the behavior described in the flow activity pattern descriptions. However, it does match the observations made in the hospital. I. e. in the real world we observed that the same patient is treated similar each day and the same caretaker is treating all patients in a similar way.

This observed behavior is effectively mimicked by the case dependent decision using the split distribution. Each decision on its own is unique and dependent on the concrete case. The course of actions taken however adheres to the range prescribed by the individual split distribution.

8.2.3. Discussion

In order to show that the flows from the generator can be used instead more flow models from other scenarios we compare their structure and the general execution behavior of both.

The workflow activity patterns have been shown to be complete [CITR08] with respect to the processes they were created from. These processes explicitly include human-intensive workflow that involve human interaction with the workflow.

The concrete process we have investigated at the hospital can also be represented based on the workflow activity patterns. Most of the observed activities match the uni-directional and bi-directional performative patterns. This depends on the habit of the caretaker and the capabilities and situation of the patient. Usually, the caretakers try to integrate the patients as much as possible, thus creating parallel execution from the perspective of the flow. Also the notification pattern is present in the case study, when information on examinations or medications are recorded for a patient. Some activities however do not fit into the patterns easily as for example the disinfect hands task. It is regularly executed whenever a caretaker is switching between two patients especially when performing the morning hygiene (mandatory activity). But these activities (or a pattern of them) can be represented using constraint based activities.

As we can basically reconstruct the structure of the real-world flow from Mainkofen with the workflow activity patterns, we conclude that the generated flows have a similar structure. This conclusion is further supported by the fact that the patterns themselves have been derived from a larger set of real-world examples.

For the general execution behavior of the generated workflows and the respective execution model we stated that we cannot create any semantic habit. But based on the knowledge from the case study we know variations on decisions and also the variance and accuracy values of recognized events. Therefore, the event sequence generated by an execution model has the same properties in terms of flexibility and preference compared to the preference of the single caretakers observed in the case study.

Based on these two properties we conclude, that the generated flow models are comparable to the one studied in Mainkofen. This allows us to use the generated Flow Models as valid foundation for simulation experiments to assess FlowPals mechanisms.

8.3. Simulation Setup

For performance evaluation we executed a large number of flows in our simulation environment. In order to do so, we require a flow engine component and a CMS component.

A real workflow engine is usually part of some corporate infrastructure. As it is difficult to set up this kind of environment, we tried to simplify the environment where possible, removing most of the services that are not relevant to assess the performance of FlowPal. We used a modular flexible flow engine based on a basic flow navigator, a single instance flow model and a basic CE. The flow engine allows further to plug in each FlowPal mechanism and compare them directly with the basic versions.

Furthermore we did abstract from a real CMS as strongly as possible in order to have fine-grained control on the properties of context event presented to the flow engine. These properties are mainly the uncertainty given by basic accuracy of the CMS and the variance in recognition quality. To keep the CMS simple, we focused on the necessary interfaces to drive the execution of flows based on context events. The CMS used in the simulation acts as discrete event generator, where each event is received by the flow engine. Furthermore the CMS is capable to modify the recognition accuracy of each event of a given context event sequence.

The required input for the simulation is created by the workflow generator. It provides a sufficient number of structurally different flow models and the respective execution models can create valid sequences of context events. We have already discussed the validity of this approach.

In the next section we describe the performance metrics that we use to assess the actual performance of the mechanisms. After that we describe the system parameters that we used to adjust the simulation environment in order to challenge the proposed algorithms of FlowPal.

8.3.1. Performance Metrics

In order to measure the performance of the mechanisms that FlowPal proposes we introduce four performance metrics. Each of them focuses on different aspects that are relevant to complete a flow execution successfully. First, the *event change rate* captures the changes ⋆Con makes on a single event instance. Second, we use the *flow certainty* to estimate the confidence of the flow engine, when making navigation decisions. Third, we investigate the overall *flow completion rate* for a given experimental setup. The flow completion rate is the most important performance

metric as it directly shows the frequency of direct human interaction required to possibly correct some executed flow. Finally, we use the *correct event assignment rate* to measure the effectiveness of FEvA when a flow system has to deal with the event sequence failures we described(cf. Section 7.2.3). In the following, we will introduce each metric in detail.

Event Change Rate The *event change rate* measures the change to the correct event type that FlowCon or FlexCon apply. This way it can show whether the application of either algorithm provides a benefit. To compute it we measure the probability of the correct event type (i. e. the one from the real-world sequence S) before (p) and after (p'') the processing of the corresponding event instance by the algorithm (cf. Section 6.3). When we measure $p = I_E^e(u)$ we preserve the original probability for u. When the probability has been modified, we record $p'' = I_E''^e(u)$ after ⋆Con has finished but before the event instance e is processed by the CE. The event change rate is defined as $ecr = \frac{p''}{p}$. Hence it yields the relative change of the probability of the correct event that the flow engine should recognize and process. This relative definition is used as event type sets of different cardinality would prevent a direct comparison of absolute changes.

The lowest value that the event change rate can assume is $ecr = 0.5$. When ⋆Con contributes no probability mass at all to the correct event its value can be halved at most. At the other end of the scale, there can be a very low original probability and a very high probability from ⋆Con. The lowest original probability can go as low as $p = 0.05$ for the simulation setup, while ⋆Con might contribute probabilities of about 0.9. This allows for event change rate values of $ecr = 9.5$ e. g. for $p = 0.05, p'' = 0.475$.

The event change rate is only used to compare FlowCon against FlexCon independently of the used CE. For the variants of HyperFlowCon the amount of information we use from ⋆Con is adapted dynamically. If we would also use the event change rate on a variant of HyperFlowCon the results would not be comparable. Therefore, the relation is different for each event and the event change rate does not capture the sole benefit of ⋆Con for experimental setups using HyperFlowCon.

Flow Certainty The *flow certainty* is the metric used to measure and compare also the effect of the CE on the result of the flow execution. When the event instances for a condition are processed by the CE the result is a single belief value. If this value is higher than the navigation threshold, the flow engine acts accordingly and the condition becomes true (cf. definition 5.2.4). The average of the belief values for all conditions of a flow is the flow certainty. That means the resulting average value represents the average confidence of the flow engine with respect

to its navigation decisions. Please note that this metric includes the effects of any event processing by ⋆Con as well as the CE.

Let $\mathcal{F}$ be a flow model with the set of conditions $C = \{c_j\}$. When we execute the flow using the event sequence $\mathcal{S}$ each event type u_i in all conditions c_j gets the belief value assigned that is the result after processing by ⋆Con denoted as $I_E''^e(u)$. That means each u_i is replaced by its final belief value and the condition evaluated $c[u_i/I_E''^e(u_i)] = v_j$. Hence, we get a single belief value for each condition, that we call transition belief v. When the transition belief is higher than the navigation threshold t_n, the respective transition is also activated. The higher the belief value, the more certain the flow engine is to navigate the flow using this transition. Therefore, we define flow certainty fc as the overall average transition belief for an executed flow instance.

$$fc = \sum_{i=1}^{j} \frac{1}{j} v_i \tag{8.2}$$

The range of the flow certainty reaches from 0.0 indicating the very unlikely situation that there was no evidence at all for any condition evaluated by the flow engine to 1.0 where for every evaluated condition there has been complete certainty.

Flow Completion Rate For all evaluated mechanisms the most important metric is the actual flow completion rate fcr. The flow completion rate is measured for a given experiment setup. In each experiment setup we execute a single flow model with k different event sequences, where k is also used as reference for the flow completion rate. Further, let s denote the number of flow instances of $\mathcal{F}$ that have been successfully executed $\mathrm{ENG}((I)(\mathcal{F}), S) = true$ in that experiment. It is then defined as $fcr = \frac{s}{k}$. This step is repeated for all available flow models executed under the same conditions.

By definition the flow completion can range from 0.0 to 1.0. The lower value denotes, that not a single flow completed its execution successfully for that experimental setup. The higher value means that every flow could be successfully executed in that experiment. Please note, that we used a large number of different flow models for each data point in our evaluation. Hence, the shown flow completion rates represent the average result for all executed flow models of a certain simulation setup.

Correct Event Assignment Rate The final metric we need is the correct event assignment rate *cear*. This metric is used for the evaluation of FEvA to assess

whether the right events have been processed and consumed by the correct activities. The generated event sequence S from the execution model is analyzed for each executed flow instance. Using knowledge from the simulation data, we know the correct activities that should process the respective events. This yields pairs of correct events and activities (a, e). When the flow instance has been executed we compare the information from the flow trace (T) with the pairs containing the correct assignment and count the correct ones as $\bar{S}$, where a correct assignment is given by $\tau(\mathcal{T}, a, u) = e$ for the pair (a, e). The correct event assignment rate is then defined as $cear = \frac{\bar{S}}{|S|}$, where $|S|$ denotes the overall number of events in the sequence.

The correct event assignment rate has a range of 0.0 to 1.0 where the lower value shows that no event has been processed as intended and 1.0 that all events have been processed by the correct activity.

8.3.2. System Parameters

The system parameters we use, can be divided in two distinct groups. The first one are the parameters used to generate the flows and control the behavior or the flow engine during the execution. The ones for creating flows were mentioned earlier in Section 8.2.1. While we already described their use, we will provide the absolute numbers used in the simulations in this section. The second group controls the CMS component of the simulation, adjusting the properties of single context events and also event sequences.

Flow-based Parameters

The first flow generation parameter is the *flow size*, that we target when generating the flow. As we have discussed, the size is given by an upper and lower activity bound and the flow generator fulfills these bounds. We choose the flow size for all experiments between 30 and 50 activities as lower and upper activity bound limit. Generated flows of this size have the most similarities in terms of activity count when compared to the flow monitored in the Mainkofen scenario. Flows of this size contain 4 to 5 patterns on average and hence provide a wide variability in possible structures. The flow size also determines the number of context events a flow must process during its execution and so with larger flows also the event sequence length and the flow history grows. As the effort for training larger flow traces (offline) increases, we did not generate flows of larger size. This is valid because all variants of ⋆Con are independent of the flow size and only depend on the complexity of the flow model (e. g. incoming transitions of an activity). Therefore, there is only a small risk for unexpected results given larger flow sizes.

The second and most important factor for the flow generation are of course the relative pattern frequencies and the *co-occurence distributions*. We did not vary this parameter in order to not break the chain of argument for the comparison to the real-world flow example. We have already shown the overall reference frequency distribution in Figure 8.13. The values for the used co-occurence probabilities can be found in Table 8.1. These values closely match the ones published by Thom et al. [TRCI08]. Some missing probabilities had to be filled in, as not all were available and the original authors of the study could/would not provide them.

Another parameter that is relevant for the simulation is the *navigation threshold* we defined in Chapter 5 (cf. definition 5.2.4 and 5.2.6). The navigation threshold is the minimum belief that the flow engine must have before it evaluates conditions based on context information and executes the the following activities. When the CMS on average provides context events with higher belief values than the navigation threshold, a stable flow execution is possible. We chose navigation thresholds between 0.4 and 0.65 for the simulation. The lower limit is given by the fact that for event type sets with only three or less event types, a lower threshold would allow kind of arbitrary choices. This is because for event instances with low significance, the actual occurrence of each event type is in the range of 0.3 or higher. The upper limit is given for a similar reason, as for event types sets with a larger number of event types, even low noise on multiple events makes it very difficult for the significant one to achieve the threshold of 0.65. We also introduced the adaptive threshold with the more sophisticated versions of HyperFlowCon where the threshold is adapted online to the significance of the event instances and the size of the event type set.

CMS parameters

The parameters that we use to control the uncertainty of a single event instance are *ground truth* and *variance*. The *ground truth GT* denotes the overall average

Table 8.1.: workflow activity pattern co-occurrence probability matrix

Co-occuring Pattern	1	2	3	4	5	6	7
1 Approval	2%	33%	30%	4%	14%	0%	17%
2 Bi-Directional	16%	13%	12%	11%	18%	5%	25%
3 Decision	26%	15%	12%	5%	28%	0%	14%
4 Informative	12%	8%	28%	16%	17%	1%	18%
5 Notification	24%	18%	28%	3%	2%	2%	23%
6 Question	30%	6%	20%	12%	15%	1%	16%
7 Uni-Dirctional	30%	10%	10%	12%	20%	5%	13%

accuracy of the CMS. I. e. on average the belief value a user of the CMS can expect is equal to GT. This represents the case when the same single event is recognized over and over again. Given a perfect CMS that always identifies a real world event correctly would lead to a ground truth $GT = 1.0$. A CMS that informs the application always with the wrong event (with perfect certainty) would have $GT = 0.0$.

In practice we have chosen values for GT between 0.4 and 0.6. These values have been used due to the results of the study in Mainkofen [KBNL11], where basic recognition rates between 14.3% and 83.3% could be achieved for individual real-world activities. On average the recognition rate was 48% and most of the activities had recognition rates between 35% and 60%. Hence, the chosen ground truth values represent the situation found in the real-world scenario closely.

While the ground truth does only represent the average recognition rate of the CMS, the individual context events are affected by the *variance* V in their recognition. We define the variance as the possible deviation range that an single event may suffer during recognition. Hence, the variance alters the average precision of the used CMS for each event instance so that it becomes $GT \pm V$. We chose a range of variance values between 0.05 for a very stable CMS and 0.75 for very unstable ones. So given a CMS with $GT = 0.5$ and $V = 0.5$, a single context event can achieve a recognition accuracy between $0.25 = GT - V/2$ and $0.75 = GT + V/2$.

Both parameters together determine the recognition quality of the CMS. Please note that this definition of ground truth is different to the usual definition in literature. We use this altered definition in order to vary the overall quality of the recognition independently of the actual situation for a single experiment.

Finally, we have the parameters for controlling the sequence errors that we introduced with the error model for FEvA (cf. Section 7.2.3).

For the false positive events denoted by α, we used values between 0.0 and 1.0. While the lower value indicates that the CMS does not produce any false positive events, the higher value describes a CMS where for each "real" event there is one false positive. We did not consider more false positive events as such a CMS produces more counterproductive events that real ones.

For the out-of-order events given by γ, we chose values between 1.0 and 0.65. Again 1.0 indicates that the CMS does not produce context events with a different real world order. For lower values $1 - \gamma$ events arrives out of order, with an "offset" of at least one event. For larger values of γ the offset increases further. Again we did not chose larger values because each out of order event affects the occurrence of two context events, the one arriving early, and the then delayed one (or vice versa). Hence, with values of 0.5 basically each event in the sequence would be affected by an out-of-order shift.

For the deleted context events determined by δ, values between 0.0 and 0.25 have been used. The value of 0.0 means no events are missed and the sequence contains all events required to execute the flow. The higher limit indicates that about one quarter of the events in the sequence are missing. As FEvA targets to resolve issues with only a single missing event it becomes impossible and also questionable to mitigate problems with multiple missing events as they can occur more frequently for larger values.

9. Simulation Results

In this chapter we present our simulation results for each of the four performance metrics we introduced. First we show that ⋆Con yields an improvement on the accuracy of events compared to the basic reference system based on the event change rate. Then we show and discuss the achievable flow completion rates for the ⋆Con variants and changes to the CE. After that, we will put the results of the advanced techniques used in HyperFlowCon and FlexCon into perspective with respect to the flow certainty metric. Finally, we conclude the chapter with the performance results of FEvA against false positive (α), out-of-order (γ) and missing events (δ).

9.1. Event Change Rate

The event change rate shows to what degree ⋆Con can make the recognition of a single context event more accurate, when the event is processed in the context the executed flow. First, we show an early result from this thesis, where we investigated the basic conditions that are required by FlowCon in order to contribute to the accuracy when executing a flow. Second, we compare the event change rate that FlexCon achieves, based on the more flexible flows modeled with the HFM in comparison against FlowCon and the more rigid flows based on the basic imperative flow model.

FlowCon We used a flow engine with FlowCon ENG(f, S)[FC], for the first evaluation. We created 100 structurally different flow models with the workflow generator. These flow models adhere to the basic imperative flow model and do not contain any constraints. Further we generated 200 individual event sequences for each flow model. The flows have been executed with each of these sequences. We always started with FlowCon freshly initialized and no training data available in the flow history $\mathcal{H}$ 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 until the learning frame is full. So in total we executed 20.000 flows for each data point in our results. We chose values for $GT = 0.45$ as suggested by the case study results. The used navigation threshold values are in a similar

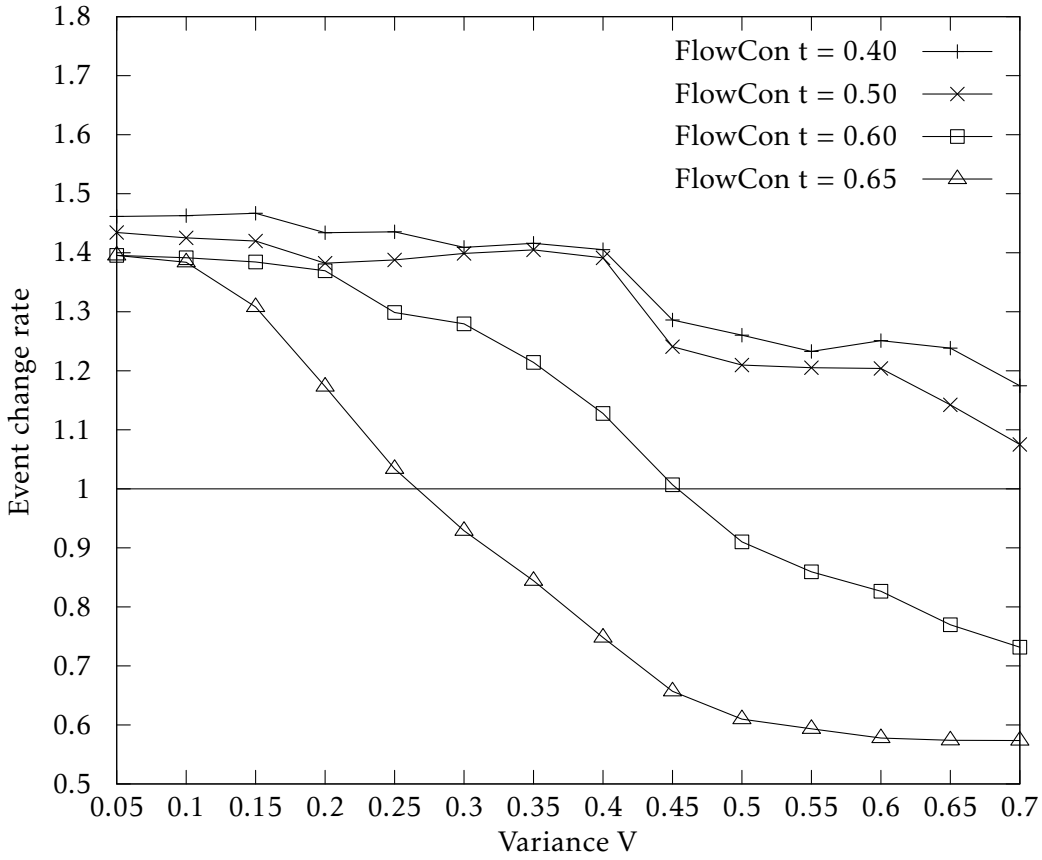

Figure 9.1.: event change rate of FlowCon for a changing navigation threshold t_n

range starting from $t_n = 0.4$ up to $t_n = 0.65$. This way given the lowest t_n of 0.4 and a low noise CMS ($V \leq 0.05$), the basic flow engine is capable finishing most of the flows successfully. This however becomes very challenging for increased navigation thresholds.

We selected values for the variance V of the CMS ranging from $V = 0.05$ and a very stable recognition (probability of the correct event 0.45 ± 0.025) to a very unstable one with $V = 0.7$ (probability of the correct event in 0.45 ± 0.375). Please note that a variance value of $V = 0.4$ basically introduces the same amount of noise in the distribution that is already present given by the ground truth $GT = 0.45$. When we 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. As a result of this unstable recognition there are some events which have much better than average recognition rates, but also the opposite is true.

The results of this simulation setup are depicted in Figure 9.1. 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 $v = 0.4$. These conditions indicate a CMS that usually meets the requirements of the flow engine. Furthermore

we can deal with a significant amount of noise quite well. However when we 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 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 for an increasing 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.

In conclusion the use of FlowCon is beneficial, when the threshold of the flow engine is not exceeding the average recognition rate of the CMS, and the variance is not the most controlling influence. Given these conditions, FlowCon is capable to learn the behavioral habit of the human involved in the process. So FlowCon significantly improves the event accuracy and aids in the correct and robust recognition of context events in imperative flows. But if the conditions for the CMS are not met, FlowCon can be beneficial. When the human user corrects the context of the flow explicitly, this leads to correct flow traces that can be used for training of FlowCon and later usage.

FlexCon We also performed the experiment with FlexCon, In order to compare the performance to FlowCon in terms of event change rate. The flows for FlowCon are purely imperative, i. e. activities are connected by transitions and there are no constraints. Thus, the task of FlowCon is much easier than that of FlexCon which also has to deal with constraints that leaves the decision about the ordering of some activities completely to the user.

We used the study results for a comparative setup given similar conditions. So for the FlexCon experiment we assume a navigation threshold $t_n = 0.4$ and a $GT = 0.45$. The variance V is also in the same range between $V = 0.05$ to $V = 0.75$. We used the extended flow generator and came up with 200 structurally different flow models with 25 to 50 activities containing also constraints and respective constraint-based activities. Again 200 traces have been executed per flow and the learning phase of the DBN is also included.

In Figure 9.2 we show the results of this comparison. 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

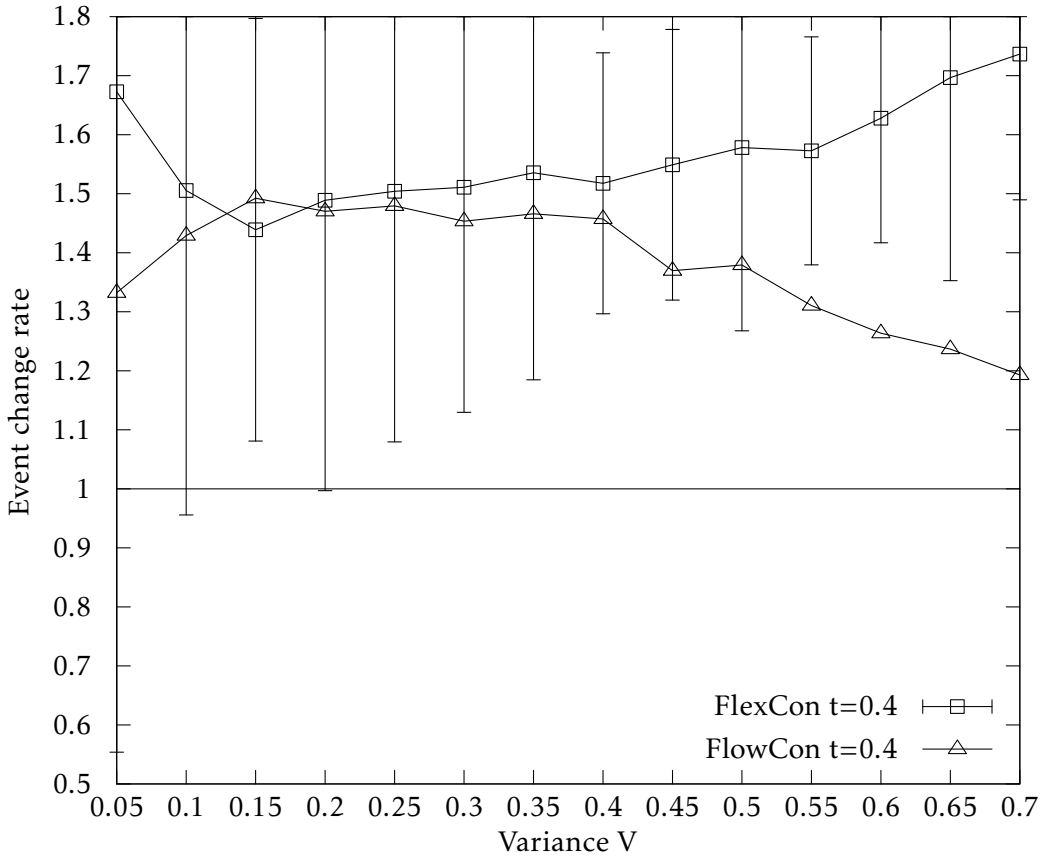

Figure 9.2.: Comparison of event change rate for FlexCon and FlowCon

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 $ecr < 1.0$ making the result worse.

The high standard deviation for the event change rate 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. Due to the fixed combination of information in this setup we end up with a $ecr = 0.5$ for those events. Using a CE with adaptive information combination reduces this problem. So according to the flow structure, we have a very high improvement for the dependent events but none for the independent ones. Overall we achive a decent improvement.

9.2. Flow Completion Rate

The flow completion rate as performance metric indicates the overall robustness of the used flow engine variant. As the metric is always calculated in reference to the overall number of flows executed per experiment, an optimal system would be achieved, when we would reach $fcr = 1.0$. In this case every flow could be executed successfully without the need for user interaction to correct some events. For each result the same assumptions as before are valid. We generated 200 structurally different flow models with the respective flow generator either with the basic imperative model for FlowCon or the HFM for FlexCon. For each model 200 traces have been executed. As before, the learning of the respective data structure in ⋆Con is included in the experiments. Hence, we do not achieve the result of $fcr = 1.0$. But given the learning frame window of 20 flow instances any $fcr \geq 0.9$ can be considered a stable configuration where every other sequence for the given flow would also lead to a successful execution, given the sequence itself is correct.

We show all the results of FlowPal configurations as we developed improvements for the algorithms. An overview on them is depicted in Table 9.1, where each column represents a property that we modified during the evaluation. The first column denotes whether the CMS provides a full probability distribution for the event type set E as information to the flow engine or only the most significant context event type u with its associated probability. The parameter in the second column denotes the trust the flow engine has in the event from the CMS. For the basic system there is no other source, so the trust equals 100%. For the basic FlowCon where we did just average there is a fixed trust of of 50%. Given a more sophisticated CE, the trust value is also controlled by the uncertainty of the event

Table 9.1.: Simulation Result Overview: Flow Completion Rate

Fig.#	context event input	trust	nav. threshold	⋆Con	CE
9.3	distribution	100%	$t_n = 0.4$	none	BinCE
9.4	distribution	50%	$t_n = 0.4$	FlowCon	BinCE
9.5	distribution	100%	$t_n = 0.4$	FlowCon	SLCE
9.6	significant event	100%	$t_n = 0.4$	FlowCon	SLCE
9.7	distribution	100%	$t_n = 0.4$	FlowCon	4V CE
9.8	distribution	60%	$t_n = 0.4$	FlowCon	SLCE
9.9	distribution	60%	$(t_n = 1/\|E\| + 0.28)$	FlowCon	SLCE
9.10	significant event	60%	$(t_n = 1/\|E\| + 0.28)$	FlowCon	SLCE
9.11	distribution	60%	$(t_n = 1/\|E\| + 0.28)$	FlowCon	4V CE
9.12	distribution	60%	$(t_n = 1/\|E\| + 0.28)$	FlexCon	SLCE
9.13	distribution	60%	$(t_n = 1/\|E\| + 0.28)$	FlexCon	4V CE

as we explained in Section 6.6.3. So 100% trust for SLCE and 4V CE means to use the uncertainty of the event, while the 60% value indicates to use at most 60% of the information from the context event, even if its uncertainty is much lower than 40%. In the third column the used navigation threshold is given, where we switch from the default "fixed" one based on the Mainkofen Scenario of $t_n = 0.4$ to the "adaptive" one with a value of $t_n = (1/|E|) + 0.28$. The last two columns indicate the used variant of ⋆Con and the CE.

The individual results are the following. First we discuss the flow completion rate of the basic flow engine as reference system. Then, we show the results of the basic FlowCon with the simple CE. The next results are then FlowCon with the more advanced SLCE, the SLCE with only the most significant event as input and FlowCon using the 4V CE. After that, we extend the SLCE with its two improvements, the minimal event uncertainty and the adaptive navigation threshold. We discuss the results with these improvements then for SLCE with only a single event and the 4V CE. Finally, we transferred our results to FlexCon and show its performance with the SLCE and 4V CE.

All the figures with the results use the same scale and metrics for comparability. Each shows the flow completion rate (z-axis) for combinations of ground truth (y-axis) $GT = 0.4$ to $GT = 0.6$ and the variance (x-axis) $V = 0.25$ to $V = 0.75$. For the range of selections the same assumptions are valid as discussed for the event change rate.

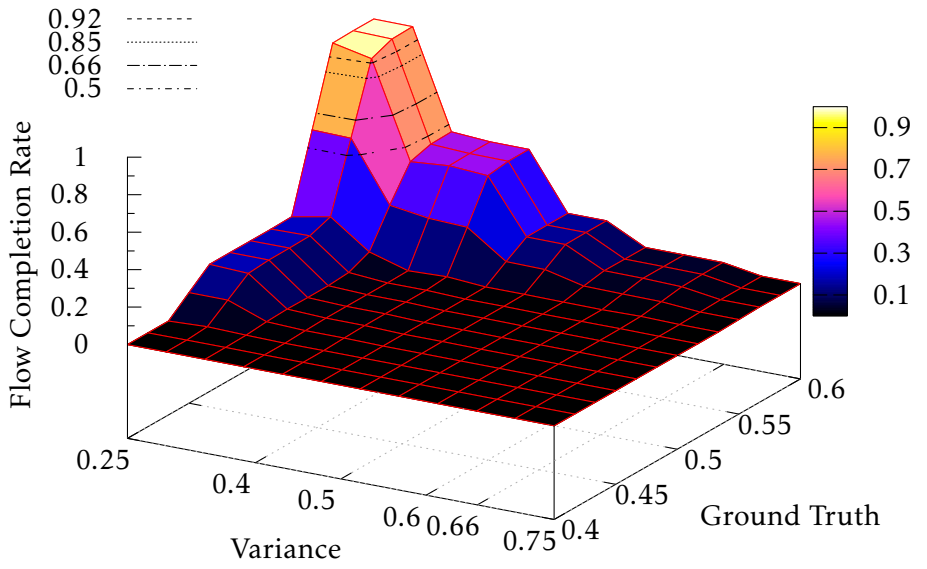

Figure 9.3.: Flow Completion Rate for basic flow engine ENG(f, S)

In Figure 9.3, we show the results of the basic reference system not using any form of FlowPal. The basic flow engine performs well ($FCR = 0.96$) for the most stable and accurate CMS ($GT = 0.6, V = 0.25$) but for all other values its performance decreases rapidly. For lower ground truth values and higher variance values at most 18% of the flows can be completed. Given the fixed navigation threshold any event with a recognition accuracy of less than 0.40 most likely breaks the flow execution and the user is required to interact with the flow directly.

FlowCon as first proposal to make the execution of flows more robust, performs much better at first glance as depicted in Figure 9.4. The respective numeric values are shown in Table 9.2, where the cells show the flow completion rate, the column header the variance and the row index the ground truth. There, we observe a major improvement for most of the variations of CMS. FlowCon is capable to aid in the completion of more than 50% of the flows for all cases and 66% for most of them ($GT \geq 0.50$). This value increases the better the recognition of the CMS becomes. However we also see that the flow completion rate becomes not as high as with the basic system, for the best CMS with $GT = 0.6$ and $V = 0.25$. There the flow completion rate is reduced from $FCR = 0.96$ to $FCR = 0.86$. This is mainly due to the training phase included in the results. Early in the experiment the BN in FlowCon does not have any information on the habit of the user executing the flow and thus does not perform well also achieving event change rates $ecr \leq 1$. As we also use a fixed rule of combination applying 50% trust to the event and using FlowCon to the same degree, the system cannot benefit fully from a highly accurate CMS.

Using SLCE we overcome the limitations of the strict combination of events and get a finer control. The results of FlowCon using SLCE are depicted in Figure 9.5 and the respective values in Table 9.3. Overall we see a significant boost of the flow completion rates for the more stable CMS between eight and ten percentage points for $GT \geq 0.55$. This improvement can be explained due to the use of the uncertainty of the event. When converted to a binominal opinion in SL we use the significance of the event instance to control the amount of information used from FlowCon. For a higher ground truth this leads to less usage of FlowCon which is

Table 9.2.: Flow Completion Rate for the basic FlowCon system

Ground Truth	Flow Completion Rate for Variance					
	0.25	0.40	0.50	0.60	0.66	0.75
$GT = 0.60$	0.86	0.83	0.81	0.78	0.77	0.75
$GT = 0.55$	0.82	0.79	0.77	0.76	0.75	0.73
$GT = 0.50$	0.77	0.75	0.74	0.72	0.71	0.68
$GT = 0.45$	0.73	0.68	0.67	0.64	0.64	0.61
$GT = 0.40$	0.58	0.56	0.56	0.56	0.56	0.54

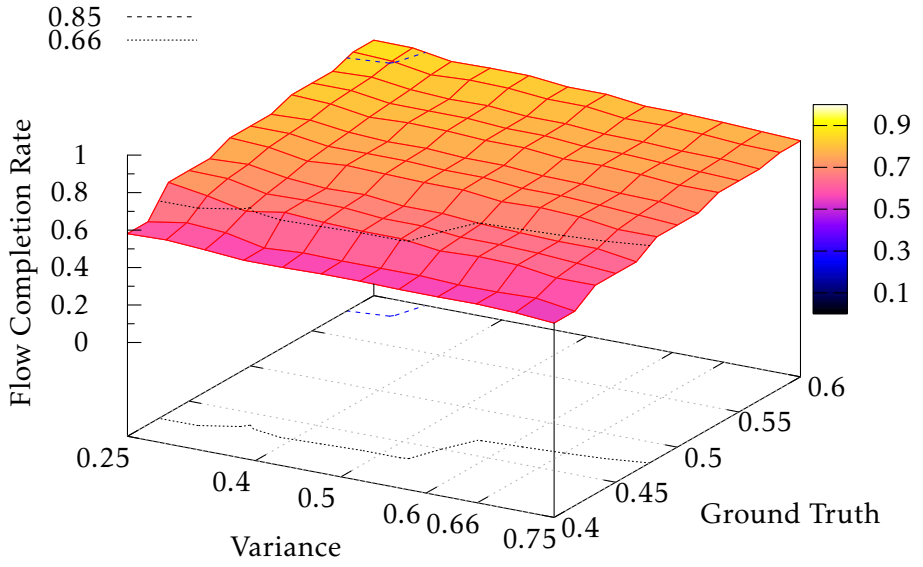

Figure 9.4.: Flow Completion Rate for the basic FlowCon system ENG(f, S)[FC]

Table 9.3.: Flow Completion Rate for FlowCon with SLCE

Ground Truth	Flow Completion Rate for Variance					
	0.25	0.40	0.50	0.60	0.66	0.75
$GT = 0.60$	0.94	0.91	0.89	0.87	0.87	0.85
$GT = 0.55$	0.89	0.86	0.86	0.84	0.82	0.81
$GT = 0.50$	0.83	0.82	0.81	0.79	0.78	0.75
$GT = 0.45$	0.79	0.77	0.75	0.74	0.70	0.64
$GT = 0.40$	0.73	0.68	0.66	0.62	0.56	0.44

desired with respect to the results of the basic system to counter the reduction of the flow completion rate. We can also benefit from this effect for low accuracy, low variance CMS ($GT = 0.45, V = 0.25$), where we achieve the highest increase of the flow completion rate of 15 percentage points. Here we have a low significance of the correct event but due to the little variance simply rely more on information from FlowCon which in turn improves the flow completion rate again. But this has also a downside as we can see in the low accuracy high variance corner of the plot, where this version actually decreases in performance again from $FCR = 0.54$ to $FCR = 0.44$. For this kind of CMS the opposite is true. A high variance leads to high significant but wrong readings by the CMS, but due to the mechanism in the SLCE, FlowPal is prone to believe in this information and uses also less information from FlowCon to counter this reading. Before we elaborate on this

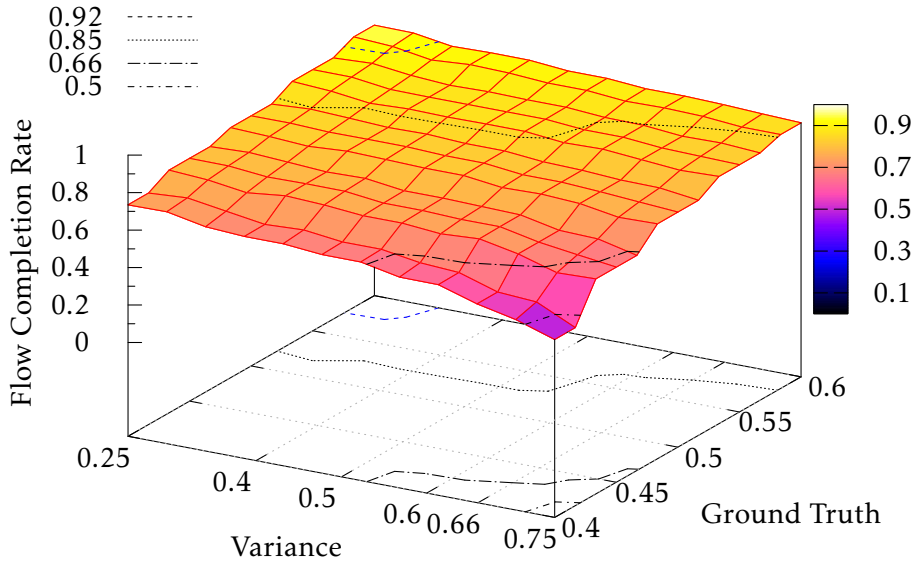

Figure 9.5.: Flow Completion Rate for FlowCon with SLCE

problem, we first look at the two alternative setups using a more sophisticated CE.

The first one uses the same set of algorithms (FlowCon with SLCE) but the CMS now only delivers the most significant event to the flow engine for processing instead of a full probability distribution on the event type set. Any probability mass not distributed on the event is then used as ignorance in the binominal opinion again, in order to control the amount of information used from FlowCon. The results of this setup are shown in Figure 9.6 and the numeric values in Table 9.4. In comparison to the previous setup having a full distribution, the single event information performs at most six percentage points worse. Indeed, having only a single event type and an overall higher uncertainty i. e. higher use of FlowCon, leads to

Table 9.4.: Flow Completion Rate for FlowCon with SLCE and the significant event type input

Ground Truth	Flow Completion Rate for Variance					
	0.25	0.40	0.50	0.60	0.66	0.75
$GT = 0.60$	0.93	0.89	0.86	0.84	0.84	0.81
$GT = 0.55$	0.87	0.84	0.83	0.80	0.80	0.78
$GT = 0.50$	0.80	0.78	0.77	0.76	0.75	0.71
$GT = 0.45$	0.75	0.73	0.72	0.70	0.68	0.63
$GT = 0.40$	0.67	0.65	0.63	0.60	0.57	0.49

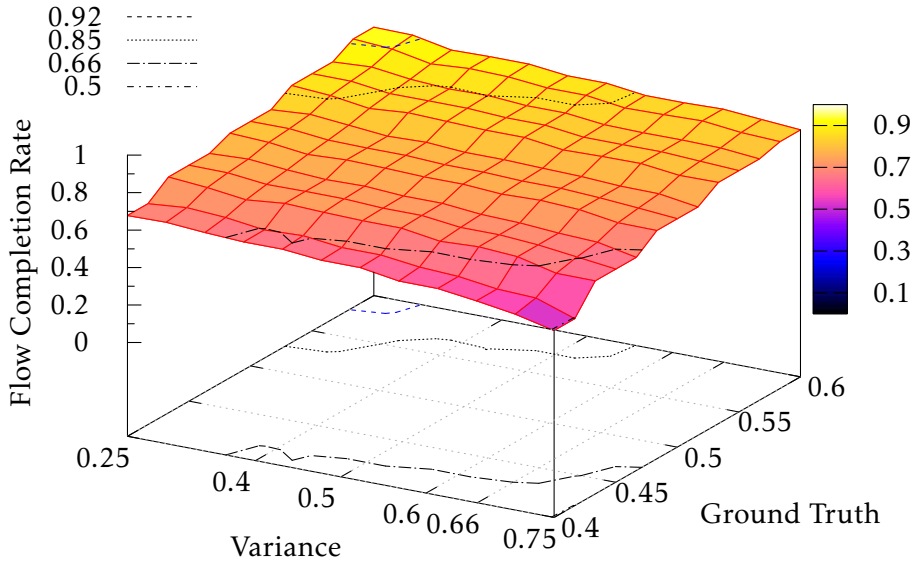

Figure 9.6.: Flow Completion Rate for FlowCon with SLCE and the significant event type input

an improved result of five percentage points for $GT = 0.4, V = 0.75$. This way the flow engine is not as prone to believe in a wrong reading.

The second variant uses FlowCon with the 4V CE (cf. Table 9.5 and Figure 9.7). Using 4V we have values for "trueness" and "knowledge" to reason on the incoming event instances and evaluate the condition. Overall 4V CE performs similar to the SLCE. For the more accurate CMS this setup performs one to two percentage points better, so using 4V is equally beneficial, compared to basic PT. However, for $GT \leq 0.45$ it is up to 34 percentage points inferior to the basic FlowCon and even 27 percentage points compared to FlowCon with SLCE. This is again due to the fact, that a high significant event leads to much less usage of information from FlowCon and hence the flow engine is again prone to believe the wrong event

Table 9.5.: Flow Completion Rate for FlowCon with 4V CE

Ground Truth	Flow Completion Rate for Variance					
	0.25	0.40	0.50	0.60	0.66	0.75
$GT = 0.60$	0.94	0.93	0.91	0.89	0.89	0.86
$GT = 0.55$	0.90	0.88	0.87	0.86	0.83	0.78
$GT = 0.50$	0.82	0.84	0.83	0.78	0.73	0.60
$GT = 0.45$	0.71	0.76	0.75	0.62	0.53	0.39
$GT = 0.40$	0.59	0.65	0.55	0.38	0.29	0.19

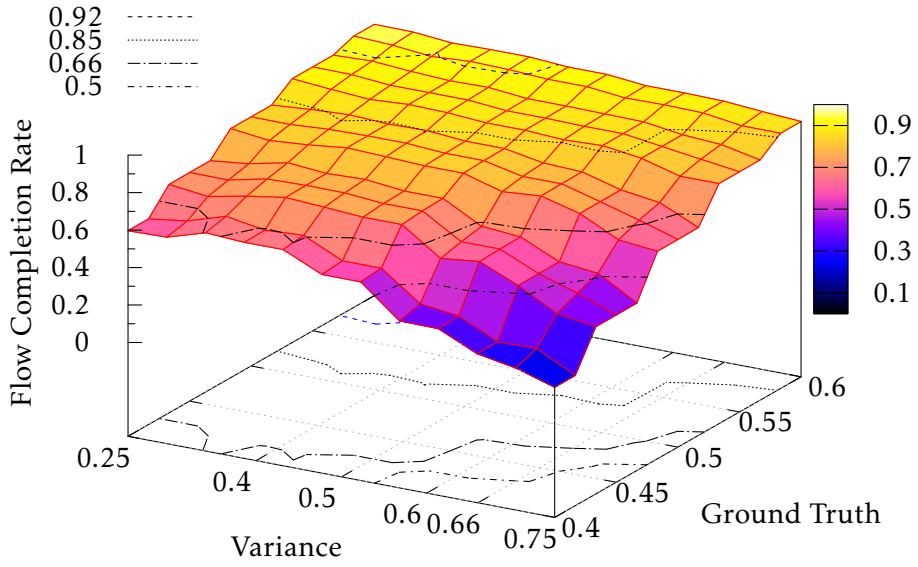

Figure 9.7.: Flow Completion Rate for FlowCon with 4V CE

happened. With only four values 4V CE is more prone to believe this type of context event. This of course reduces the flow completion rate as shown in Figure 9.7 The last results lead to the development of the improved versions of SLCE that use a fixed amount of trust, so that FlowCon (respective ⋆Con) is always applied to some degree to counter high significant but wrong events.

The results of the first version applying a trust of 60% to FlowCon with SLCE are depicted in Figure 9.8, the respective flow completion rate values in Table 9.6. The performance is basically identical to the variant with 100% trust, with a noticeable deviation for the less accurate CMS with $GT \leq 0.45$. Here we see an improvement in flow completion rate, especially for the higher variance values. The peak improvement of 18 percentage points is achieved for $GT = 0.40, V = 0.75$. So, while

Table 9.6.: Flow Completion Rate for FlowCon with SLCE and a maximal trust of 60%

Ground Truth	Flow Completion Rate for Variance					
	0.25	0.40	0.50	0.60	0.66	0.75
$GT = 0.60$	0.93	0.91	0.89	0.87	0.87	0.85
$GT = 0.55$	0.88	0.86	0.86	0.84	0.84	0.82
$GT = 0.50$	0.83	0.82	0.82	0.81	0.80	0.78
$GT = 0.45$	0.78	0.78	0.78	0.76	0.74	0.73
$GT = 0.40$	0.73	0.71	0.68	0.66	0.66	0.63

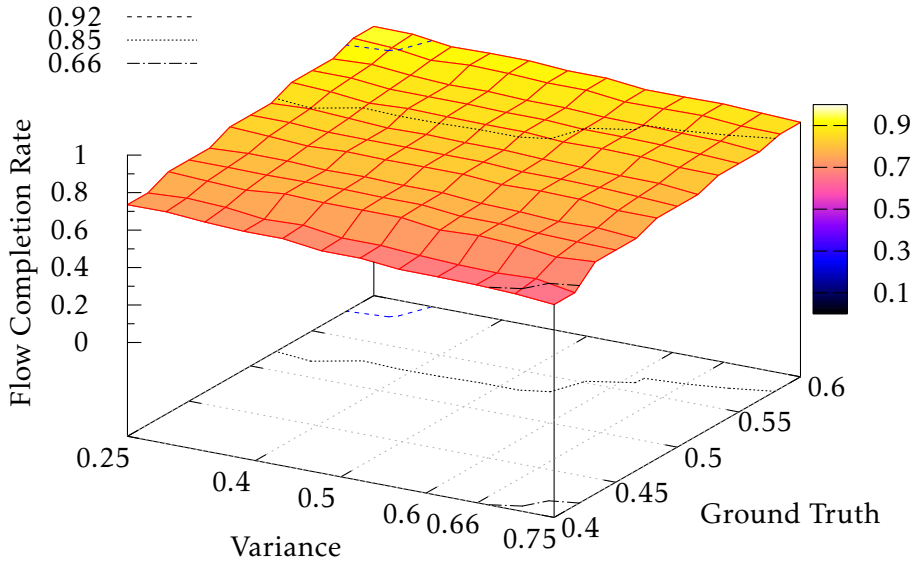

Figure 9.8.: Flow Completion Rate for FlowCon with SLCE and a maximal trust of 60%

for the more accurate CMS the use of FlowCon is balanced enough in order to not interrupt normal execution, we gain significant assistance for the worse context event.

The results of the next improvement we applied are shown in Figure 9.9 and Table 9.7. Here, the adaptive navigation threshold has been added to FlowCon with SLCE including the improved trust value of 60%. Overall, we see a significant improvement gained by the adaptive threshold. The improvements can be summarized with the following two statements. The higher the GT, the lower the benefit as the fixed threshold has already been in the correct range. The higher the variance, the higher the benefit, as the combination of the trust value and the adaptive

Table 9.7.: Flow Completion Rate for FlowCon with SLCE, a maximal trust of 60% and an adaptive navigation threshold

Ground Truth	Flow Completion Rate for Variance					
	0.25	0.40	0.50	0.60	0.66	0.75
$GT = 0.60$	0.99	0.98	0.98	0.97	0.97	0.96
$GT = 0.55$	0.97	0.97	0.96	0.95	0.95	0.94
$GT = 0.50$	0.94	0.94	0.93	0.93	0.92	0.92
$GT = 0.45$	0.88	0.89	0.90	0.89	0.89	0.88
$GT = 0.40$	0.81	0.84	0.83	0.82	0.82	0.79

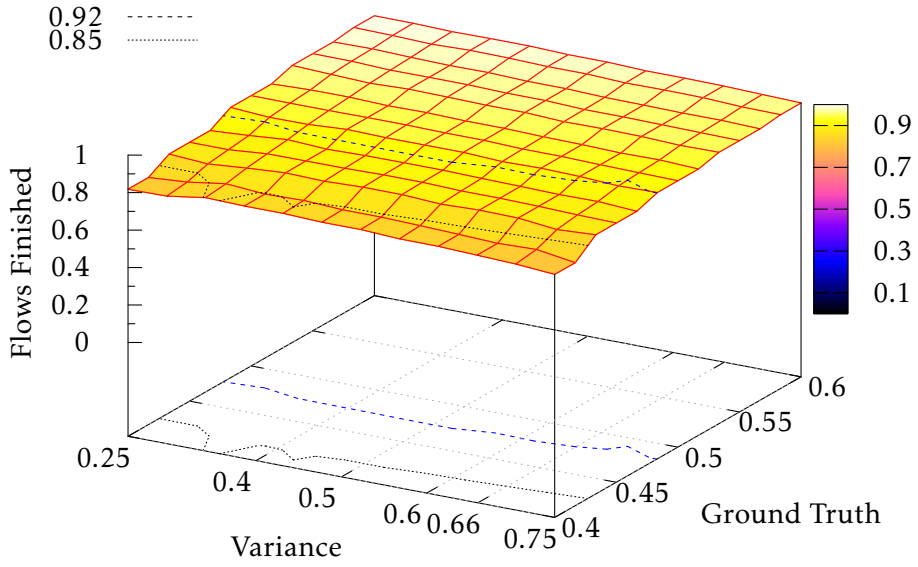

Figure 9.9.: Flow Completion Rate for FlowCon with SLCE, a maximal trust of 60% and an adaptive navigation threshold

threshold mitigate the high variance on wrong events better than the trust value alone. Hence the smallest performance increase is achieved for $GT = 0.60, V = 0.25$ where $FCR = 0.99$ went up six percentage points. The largest improvement then happens on the opposite side of the search space for $GT = 0.40, V = 0.75$ where $FCR = 0.79$ went up 16 percentage points. To put this result further into perspective we compare it against the former variants using only a single event type as input and the 4V CE.

In Figure 9.10 are the results for FlowCon with SLCE using the minimal trust, the adaptive navigation threshold and only the most significant event from the CMS as input, the respective values are in Table 9.8. On average the flow completion

Table 9.8.: Flow Completion Rate for FlowCon with SLCE, a maximal trust of 60%, an adaptive navigation threshold and the significant event type input

Ground Truth	Flow Completion Rate for Variance					
	0.25	0.40	0.50	0.60	0.66	0.75
$GT = 0.60$	0.98	0.96	0.95	0.94	0.93	0.92
$GT = 0.55$	0.95	0.93	0.91	0.90	0.90	0.88
$GT = 0.50$	0.91	0.88	0.88	0.87	0.86	0.85
$GT = 0.45$	0.83	0.83	0.83	0.82	0.82	0.81
$GT = 0.40$	0.70	0.73	0.75	0.76	0.75	0.75

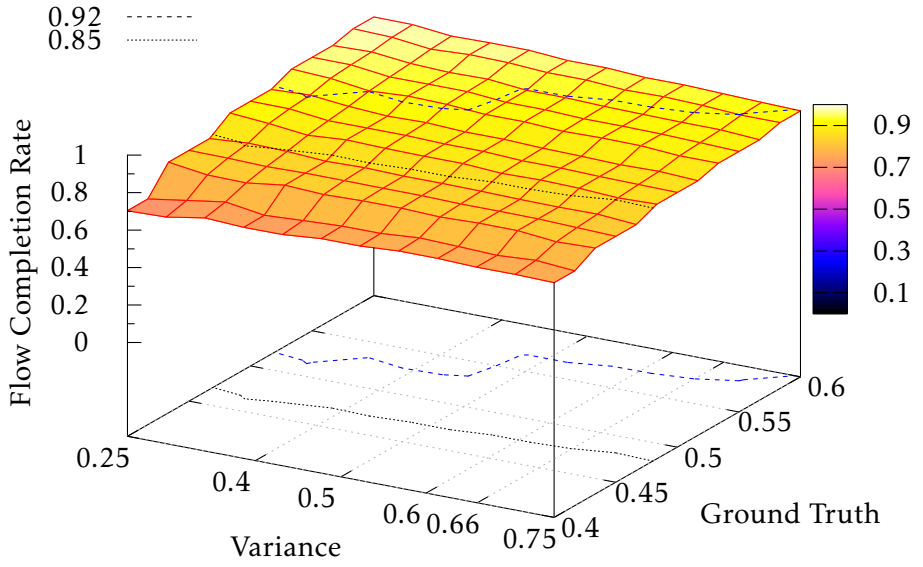

Figure 9.10.: Flow Completion Rate for FlowCon with SLCE, a maximal trust of 60%, an adaptive navigation threshold and the significant event type input

rate decreases by five percentage points, which is similar to the previous results, where we moved from the full distribution to the single most significant event. However, there is one interesting behavior for $GT = 0.40, V \leq 0.5$, where we are up to 16 percentage points worse. In this combination it can happen that we receive a wrong context event but all the mass for the right event is attributed to FlowCon. FlowCon aids in the detection of the correct event but also will improve the unlikely but possibly false event. As we have no other information from the CMS in this setup, the false event still has a fair chance to be higher than the adapted navigation threshold. Hence, the performance is decreased. If we also compare this result with the original FlowCon with SLCE and the reduced input from the CMS this improved setup is still a slight three percentage points ahead.

The final variant of FlowCon uses the 4V CE with the two improvements of 60% trust the adaptive navigation threshold. Figure 9.11 depicts the results and the values can be found in Table 9.9. The trust value and the adaptive navigation threshold have nearly no effect for the more accurate CMS. There we have only a slight improvement up to four percentage points compared to the former version using FlowCon with 4V CE. The opposite result is achieved for the high variance $V = 0.75$, low accuracy $GT \leq 0.50$ systems. There we achieve improvements in the flow completion rate between 13 and up to 48 percentage points. Here the beneficial effects that could be observed for the other systems also influence the

Table 9.9.: Flow Completion Rate for FlowCon with 4V CE, a maximal trust of 60% and an adaptive navigation threshold

Ground Truth	Flow Completion Rate for Variance					
	0.25	0.40	0.50	0.60	0.66	0.75
$GT = 0.60$	0.98	0.96	0.95	0.94	0.93	0.92
$GT = 0.55$	0.94	0.93	0.92	0.91	0.90	0.88
$GT = 0.50$	0.89	0.89	0.88	0.87	0.86	0.85
$GT = 0.45$	0.85	0.83	0.82	0.80	0.79	0.71
$GT = 0.40$	0.76	0.74	0.73	0.72	0.71	0.68

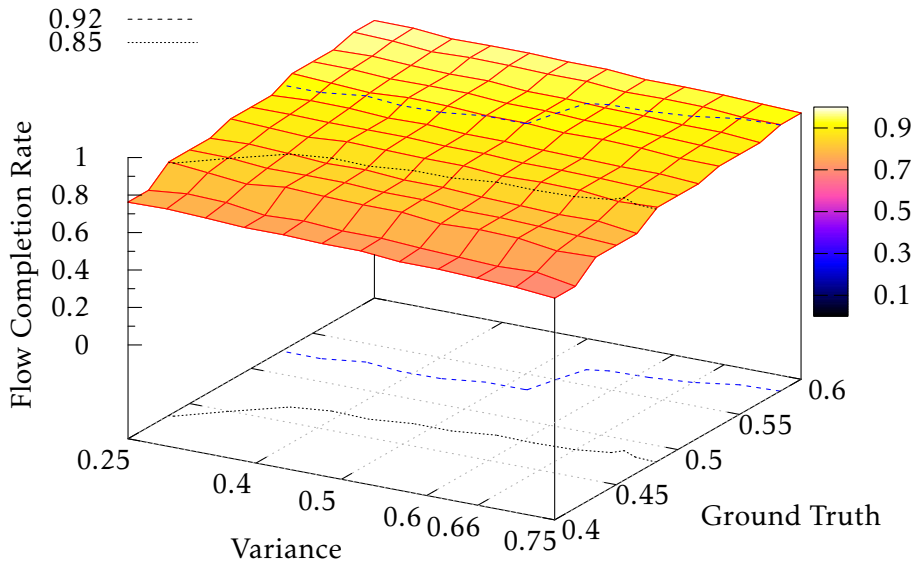

Figure 9.11.: Flow Completion Rate for FlowCon with 4V CE, a maximal trust of 60% and an adaptive navigation threshold

flow completion rate. The higher the variance, the higher the benefit, because both improvements prevent the flow engine believing a wrong respective contradictory event.

Having achieved very high flow completion rates for FlowCon we transferred the results to FlexCon. In Figure 9.12 the results of FlexCon with SLCE, the 60% trust value and the adaptive navigation threshold are shown; Table 9.10 shows the respective results. Please note again, that these results have been achieved on the much more flexible flows based on the HFM. On average we achieve a flow completion rate of 77%, which is only 14 percentage points behind FlowCon with the same adjustments but executing the more rigid flows.

Table 9.10.: Flow Completion Rate for FlexCon with SLCE, a maximal trust of 60% and an adaptive navigation threshold

Ground Truth	Flow Completion Rate for Variance					
	0.25	0.40	0.50	0.60	0.66	0.75
$GT = 0.60$	0.91	0.87	0.85	0.84	0.83	0.82
$GT = 0.55$	0.85	0.83	0.82	0.81	0.80	0.80
$GT = 0.50$	0.78	0.78	0.77	0.77	0.77	0.76
$GT = 0.45$	0.73	0.73	0.74	0.74	0.74	0.73
$GT = 0.40$	0.64	0.67	0.67	0.68	0.67	0.66

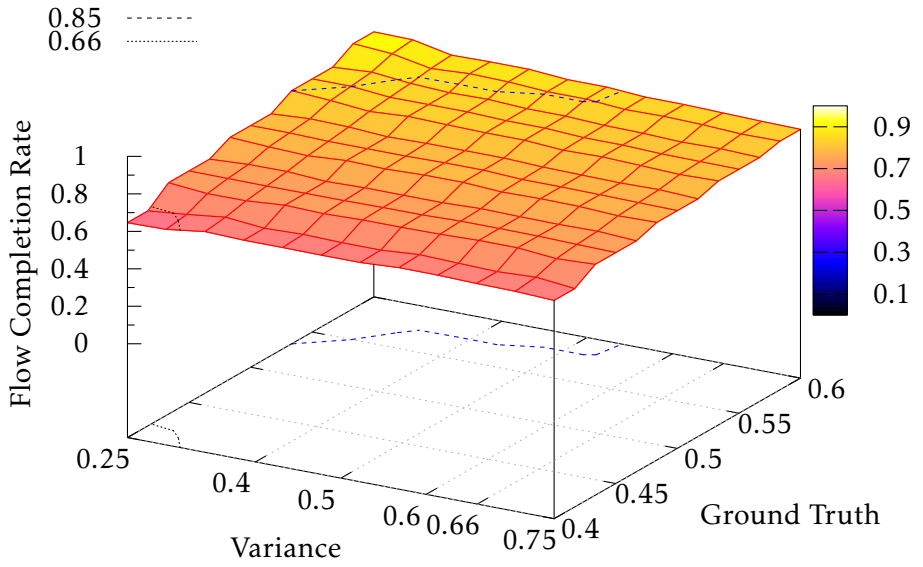

Figure 9.12.: Flow Completion Rate for FlexCon with SLCE, a maximal trust of 60% and an adaptive navigation threshold

Table 9.11.: Flow Completion Rate for FlexCon with 4V CE, a maximal trust of 60% and an adaptive navigation threshold

Ground Truth	Flow Completion Rate for Variance					
	0.25	0.40	0.50	0.60	0.66	0.75
$GT = 0.60$	0.98	0.96	0.93	0.90	0.88	0.86
$GT = 0.55$	0.95	0.91	0.88	0.85	0.83	0.80
$GT = 0.50$	0.88	0.83	0.81	0.79	0.77	0.75
$GT = 0.45$	0.77	0.74	0.72	0.70	0.70	0.67
$GT = 0.40$	0.65	0.63	0.63	0.62	0.61	0.60

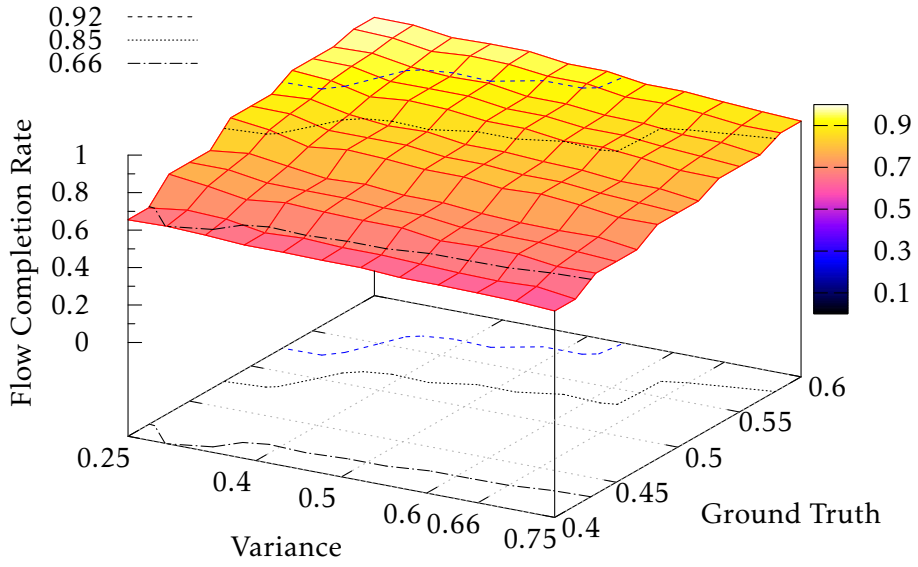

Figure 9.13.: Flow Completion Rate for FlexCon with 4V CE, a maximal trust of 60% and an adaptive navigation threshold

Using FlexCon with 4V CE yields similar results, that are depicted in Figure 9.13, and shown in Table 9.11. Compared to the other FlexCon setup we get a mixed picture. Using the 4V CE the flow completion rate is significantly better for the more accurate CMS with $GT \geq 0.50$. There, we achieve an improvement of up to ten percentage points, especially for low variance values $V \leq 0.40$. However, for the less accurate CMS with $GT \leq 0.45$ the flow completion rate drops a slight six percentage points. The increase can be explained because the 4V CE also takes contradicting information into account and is less sensitive in the condition evaluation as we will also see in the next section when we discuss the flow certainty. The drop for the less accurate CMS is caused by a combination of effects. Due to a "missing" relation in FlexCon a contradiction in 4V CE is imposed, which in turn allows to navigate one or even multiple wrong transitions, maybe in addition to the right one. This likely leads then also to a wrong execution. A higher variance supports this effect. Therefore, it can also be observed for $GT = 0.50, V = 0.75$ where 4V CE is two percentage points worse than its SLCE variant.

Discussion The flow completion rate is the key performance indicator for the usefulness of FlowPals algorithms. We have started our work with a mediocre basic flow navigator that is only capable to handle uncertain context information with the most accurate CMS. With the introduction of FlowCon we have shown that the usage of flow knowledge can provide a significant benefit in order to achieve

better flow completion rates when the CMS cannot provide very accurate results. However, we also have seen, that the naive way to use this information reduces the performance of for accurate CMS.

Spending effort to combine the information from ⋆Con in a more meaningful way using either SL CE or 4V CE has helped to improve the completion rates significantly and also allows us to reduce the amount of statistical information required from ⋆Con back to a minimum baseline. But it was also quite surprising that a fair amount of ignorance also helps to improve the flow completion rate. This way the flow becomes more resilient to high significant but wrong events. This especially has boosted the completion rates for the lower quality CMS. Our results show further a gracefully degradation of the algorithm performance the worse the CMS becomes.

Based on these combination methods we were even able to drop the requirement on having a full probability distribution from the CMS to only the most significant event, which makes the application of FlowPal much simpler to available CMS.

Having transferred this set of mechanisms to the flexible human-centric flows based on the HFM, with similar success is a very important overall result. We are capable to extract enough information from this more loosely coupled process description so that we can provide a decent second source of information to handle uncertain context information.

9.3. Flow Certainty

Now, that we have seen the overall performance ⋆Con can achieve with its different variants, we also want to show the decrease in uncertainty. As we described in Section 8.3.1, the flow certainty can be used to argue on the navigation decisions made by the flow engine. The results on the flow certainty have been measured given identical simulation setups as described for the flow completion rate i. e. 200 structurally different flow models and 200 traces for each model.

The overview on the achieved results is shown in Table 9.12. We start with the average event certainty that is fed into the system, followed by the flow certainty of the basic FlowCon system. We compare the results against the improved versions of HyperFlowCon with the adaptive threshold and the limited trust and also for the reduced input information. After that, we discuss the certainty results of FlexCon with SLCE.

All the figures with the results use the same scale and metrics for comparability. Each shows the flow certainty (z-axis) for combinations of ground truth (y-axis) $GT = 0.4$ to $GT = 0.6$ and the variance (x-axis) $V = 0.25$ to $V = 0.75$. This range

is the same used for the flow completion rate. The flow certainty is depicted in a range from $fc = 0.40$ to $fc = 1.0$

Figure 9.14 shows the average flow certainty when we use only the basic reference system without FlowCon and the "default" Condition Evaluator. This setup achieves a minimal flow certainty for the low accurate CMS $GT = 0.4, V = 0.25$ of 31% and a maximum one for high accurate CMS $GT = 0.6, V = 0.25$ of 56%. For the less accurate CMS a higher variance slightly improves the certainty as the correct events tend to benefit more from the normalizing step after applying the variance. The opposite holds for the more accurate CMS. Therefore, the flow certainty increases for the low accurate CMS with high variance $GT = 0.4, V = 0.75$ where it is 39% and decreases for the high accurate CMS with high variance $GT = 0.6, V = 0.75$ where it is only 49%. Overall the flow certainty is lower than actually expected due to the average ground truth. This is due to the loss of information when combining the events using probability theory e. g. by the use of probabilistic conjunction.

Table 9.12.: Simulation Result Overview: Flow Certainty

Fig.#	context event input	trust	nav. threshold	⋆Con	CE
9.14	distribution	100%	$t_n = 0.4$	n.a.	BinCE
9.15	distribution	50%	$t_n = 0.4$	FlowCon	BinCE
9.16	distribution	60%	$(t_n = 1/\|E\| + 0.28)$	FlowCon	SLCE
9.17	significant event	60%	$(t_n = 1/\|E\| + 0.28)$	FlowCon	SLCE
9.18	distribution	60%	$(t_n = 1/\|E\| + 0.28)$	FlexCon	SLCE

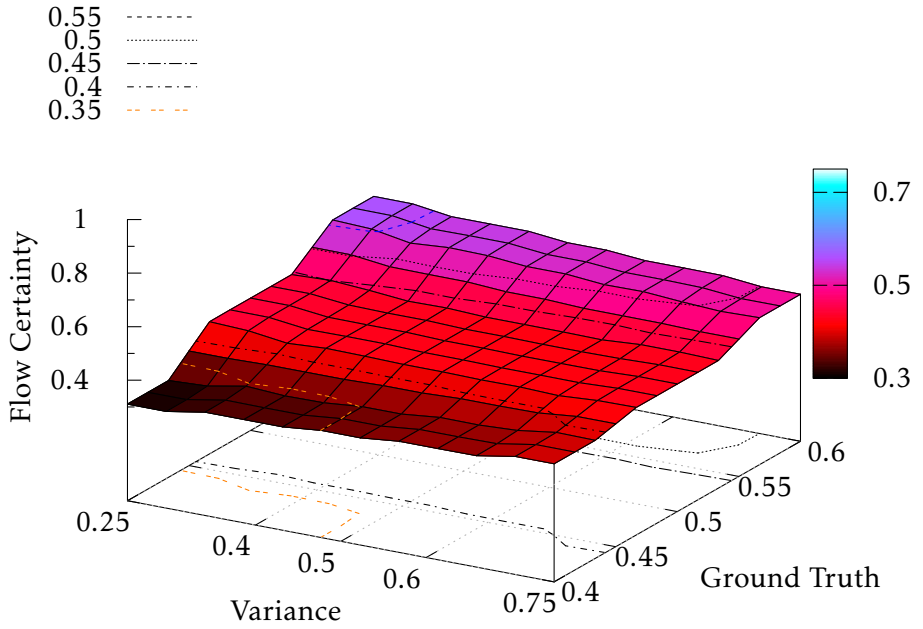

Figure 9.14.: Average Flow Certainty of the basic reference System

Next, we compare those results with the improvements that FlowCon can achieve. The results are depicted in Figure 9.15 and the respective data points in Table 9.13. FlowCon manages to decrease the effects of high variance, especially for the lower accurate CMS where the variance can be tolerated very well. There we gain up to 12 percentage points compared to the reference system without FlowCon. For the more accurate CMS, the overall flow certainty is reduced by about two percentage points. Here the fixed combination of information used by basic FlowCon prohibits a better result. We have already shown and discussed this effect for the flow completion rate (cf. Figure 9.4).

When we apply the final set of algorithms devolved for FlowCon including SLCE, the maximal trust of 60% and the adaptive navigation threshold, we achieve flow

Table 9.13.: Flow Certainty for the basic FlowCon system

Ground Truth	Flow Certainty for Variance					
	0.25	0.40	0.50	0.60	0.66	0.75
$GT = 0.60$	0.54	0.51	0.50	0.49	0.49	0.48
$GT = 0.55$	0.50	0.48	0.47	0.47	0.47	0.47
$GT = 0.50$	0.46	0.45	0.45	0.45	0.45	0.45
$GT = 0.45$	0.44	0.44	0.44	0.44	0.44	0.45
$GT = 0.40$	0.44	0.44	0.44	0.44	0.44	0.44

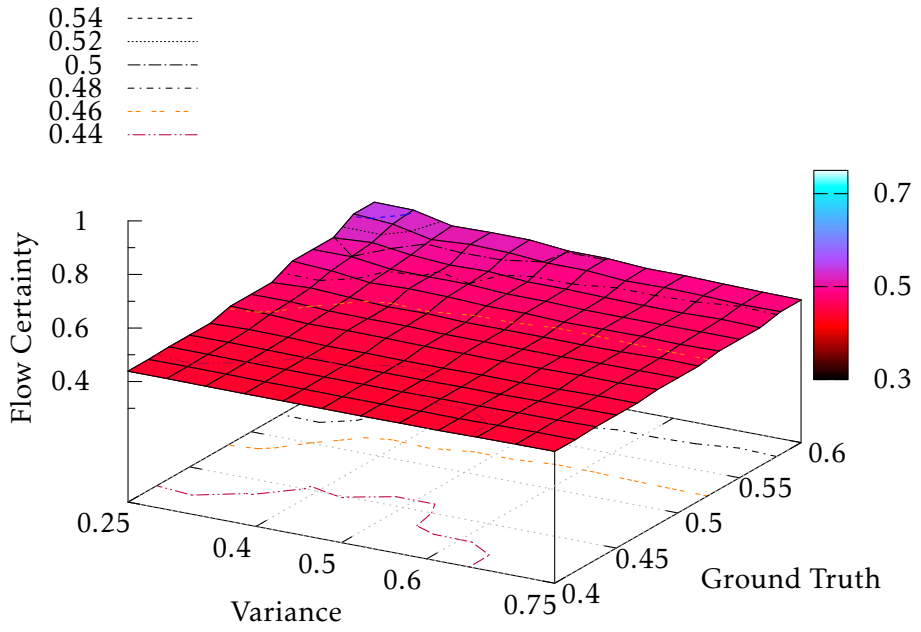

Figure 9.15.: Flow Certainty for the basic FlowCon system

certainty results as depicted in Figure 9.16. The respective values are printed in Table 9.14. On average over all measurement points we achieve a flow certainty of 64%. Please note that there are two local maximum values in the plot. One for the most accurate CMS $GT = 0.6, V = 0.25$ with $fc = 0.65$ and the other for the worst CMS $GT = 0.4, V = 0.75$ also with $fc = 0.65$. Using the optimized algorithm, allows for precise adjustment to which degree we rely on FlowCon for information provisioning. For the better CMS, we limit the use of FlowCon to a degree necessary to deal with the uncertainty introduced by the increasing variance. Therefore, the results achieve values as could be expected by the ground truth alone. However, for the worse CMS we also achieve very robust results. Of course, we cannot magically remove the uncertainty introduced by the CMS. But in this cases the flow engine relies more on the statistical information from previous flow executions that can be provided by FlowCon. This allows us also to get a high flow certainty at the expense not to use the real world events to a large degree. A higher variance – i. e. more uncertainty – again favors to use more information from FlowCon compared to the more accurate CMS. Therefore FlowCon with SLCE tries to remove the uncertainty and in turn uses even less information from the context event. Of course, this introduces a risk to take a decision purely on statistical data than based on evidence. But as the flow provides still some structure, a single error will usually be detected for one of the following events, which allows fast correction. This provides also an opportunity for FEvA to aid in successful flow execution (i. e. a combination of a missing and a false positive event).

Table 9.14.: Flow Certainty for FlowCon with SLCE, a maximal trust of 60% and an adaptive navigation threshold

Ground Truth	Flow Certainty for Variance					
	0.25	0.40	0.50	0.60	0.66	0.75
$GT = 0.60$	0.65	0.64	0.63	0.62	0.62	0.62
$GT = 0.55$	0.64	0.63	0.62	0.62	0.62	0.62
$GT = 0.50$	0.61	0.62	0.62	0.62	0.62	0.63
$GT = 0.45$	0.60	0.62	0.62	0.63	0.63	0.64
$GT = 0.40$	0.62	0.63	0.64	0.64	0.65	0.65

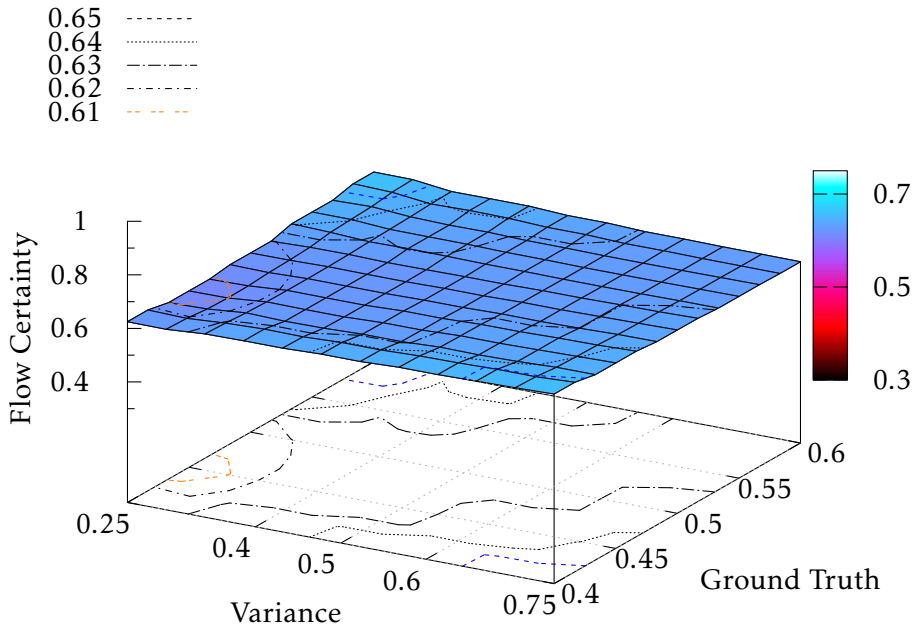

Figure 9.16.: Flow Certainty for the FlowCon with SLCE, a maximal trust of 60% and an adaptive navigation threshold

In the next step, we reduce the input information again providing only the most significant event. The results are shown in Figure 9.17 and Table 9.15. For the more accurate CMS the result is only three to nine percentage points worse than before. For these situations we receive the correct event most of the time and with the significance based usage of FlowCon, the results remain similar. For the less accurate CMS the drop in of flow certainty is greater. Here the CMS can provide no evidence at all for the correct event and even using a possibly high amount of uncertainty going along with the event on average limits the flow certainty. This is also sound with the observations made for the flow completion rate as depicted in Figure 9.10.

Table 9.15.: Flow Certainty for FlowCon with SLCE, a maximal trust of 60% and an adaptive navigation threshold and reduced input information

Ground Truth	Flow Certainty for Variance					
	0.25	0.40	0.50	0.60	0.66	0.75
$GT = 0.60$	0.63	0.61	0.59	0.58	0.57	0.56
$GT = 0.55$	0.60	0.58	0.57	0.56	0.55	0.55
$GT = 0.50$	0.56	0.55	0.54	0.54	0.54	0.54
$GT = 0.45$	0.52	0.52	0.52	0.53	0.53	0.53
$GT = 0.40$	0.48	0.51	0.51	0.52	0.53	0.53

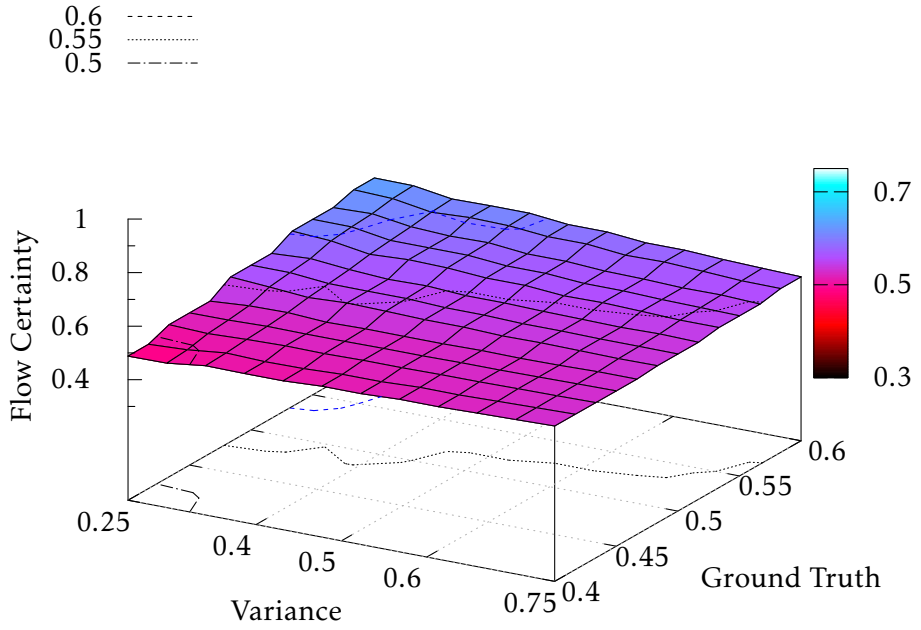

Figure 9.17.: Flow Certainty for the FlowCon with SLCE, a maximal trust of 60% and an adaptive navigation threshold and reduced input information

Finally, Figure 9.18 shows the results on flow certainty of FlexCon applied to hybrid flows. The respective values can be found in Table 9.16. Overall the results in terms of flow certainty are comparable to those achieved with FlowCon for the imperative flows. Again, for the accurate CMS a viable certainty of up to 61% is achieved. This means that the FlexCon is also capable to provide the right amount of information for the less structured hybrid flows. But also for the less accurate CMS we have a similar issue as FlowCon. On average, we have the highest flow certainty for the worst CMS due to the fact, that the high uncertainty leads to a stronger influence from FlexCon. As we then use less of the uncertain information, the flow engine is sensitive to uncommon execution sequences where the user deviates from his "routine" behavior, to a degree that is not reflected by

FlexCon. An over-trained DBN will increase this issue, as the routine case can have an over proportional weight. Similar to FlowCon this is somewhat mitigated by the fact, that an event which does not fit in the current routine, will likely cause the flow engine to ask for user guidance quickly. Given the right event, FlexCon will correctly recover and represent the statistics of the correct real-world situation, providing further assistance. Analyzing the single executions of this setup showed that this kind of interaction would have been required in less than 4% of the executed flows. This value seems acceptable.

Discussion The results on the flow certainty show that our methods are capable for a broad range of CMS to improve the confidence of the flow when making navigation decisions. This is especially true for the more accurate CMS but those posed only a small problem right from the start. On the other end, for the more inaccurate CMS we observe a mixed result. The flow certainty in general is very high but this raises suspicion. When combined with SLCE or 4V CE and a very inaccurate CMS, the flow engine is prone for two kinds of erroneous behavior. Either it uses a very high amount of information from ⋆Con due to the lack of information from the received context events and thus may omit information that deviate from the statistical most likely execution path in the flow. The other case happens when the CMS due to its high variance recognizes a wrong event with high significance. In this case FlowPal is prone to belief in this event and reduce the amount of information used by ⋆Con to counter it. Both situations will quickly require the attention of the user, as subsequent information will soon contradict with the previous navigation decision. This will negatively impact the unobtrusive execution of the flow.

Table 9.16.: Flow Certainty for FlexCon with SLCE, a maximal trust of 60% and an adaptive navigation threshold

Ground Truth	Flow Certainty for Variance					
	0.25	0.40	0.50	0.60	0.66	0.75
$GT = 0.60$	0.61	0.60	0.59	0.58	0.58	0.58
$GT = 0.55$	0.60	0.59	0.58	0.58	0.58	0.59
$GT = 0.50$	0.58	0.58	0.59	0.59	0.60	0.61
$GT = 0.45$	0.58	0.59	0.60	0.61	0.61	0.62
$GT = 0.40$	0.60	0.61	0.62	0.64	0.64	0.65

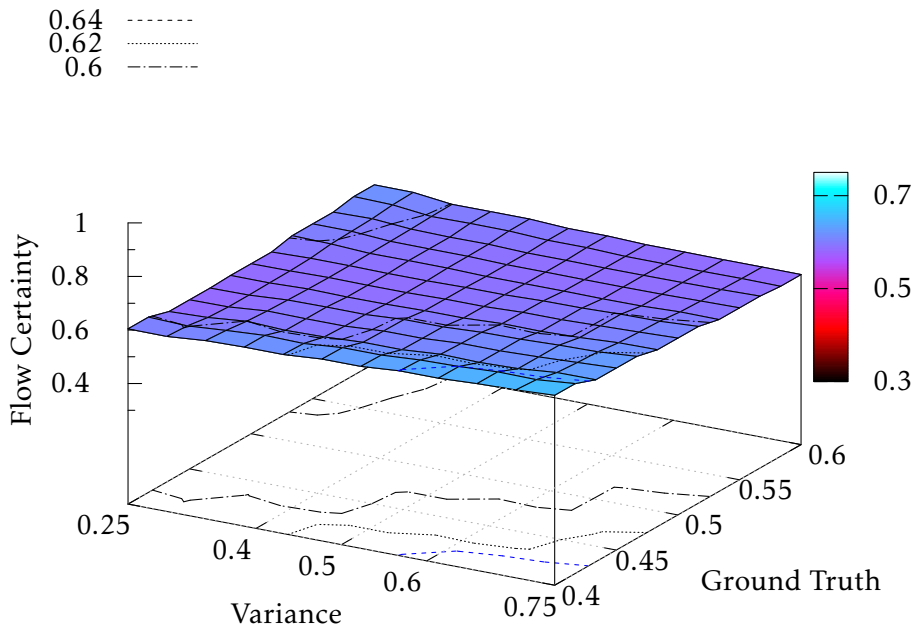

Figure 9.18.: Flow Certainty for FlexCon with SLCE, a maximal trust of 60% and an adaptive navigation threshold

9.4. FEvA results

In the final part of this chapter we show the detailed results achieved by FEvA. First we will investigate the performance of FEvA with respect to the correct event assignment rate for all three parameters α, γ, δ of the error model. Each parameter is discussed independent of the others. After that we discuss the associated flow completion rates that can be achieved. Finally, we directly compare the basic flow engine ENG(f, S) against FEvAs flow engine ENG(f, S)[FEvA].

9.4.1. Correct Event Assignment Rate

For the simulations here we used again the set of 200 different flow models and the respective 200 trace per model. The chosen range of parameter values for α, γ, δ have already been discussed in Section 8.3.1. For the experiments with FEvA we have set the ground truth to $GT = 0.5$ as this value matches the minimal linguistic requirement "high" for FEvA to accept an event and assign it. As we do not include ⋆Con, which also affects the accuracy of the events, for these experiments, this assumption is reasonable. On average an generated event will achieve the desired accuracy. Further, we investigate three different variance values $V \in 0.2, 0.4, 0.5$ for CMS with low, medium and medium-high noise.

In each diagram we use stacked histograms to show the correct event assignment rate (y-axis). In the foreground (dark grey) we see the results of the basic reference flow engine ENG(f, S). On top of these bars are the results for an flow engine using FEvA. On the x-axis the value for the respective error model parameter is given. The range of values and their validity have already been discussed at the end of Section 8.3.2. Additionally, the diagrams are grouped according to the variance value used for the experiment.

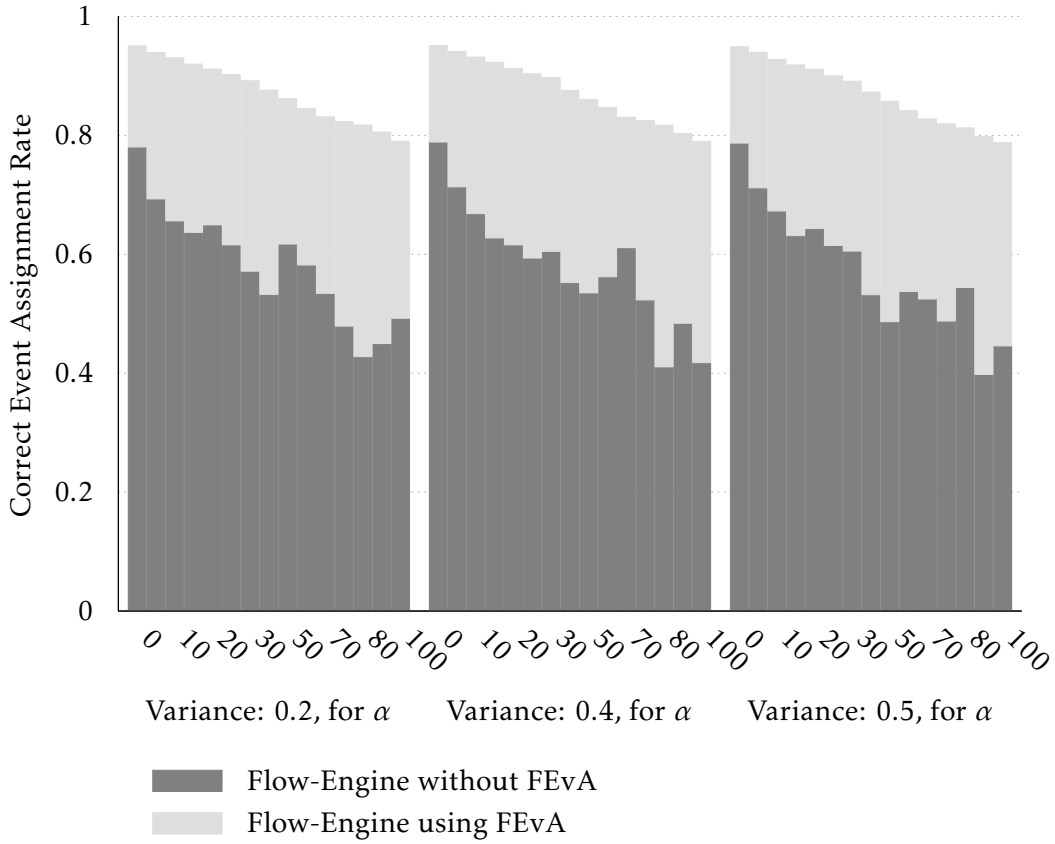

Figure 9.19.: Correct Event Assignment Rate comparison of reference system and FEvA for false positive context events

Figure 9.19 shows the results for false positive events α. The number of correct events assigned to the activities decreases slowly from about $cear = 0.94$ to about $cear = 0.78$, for an increasing amount of false positive events. This is a solid result and indicates that FEvA is capable to assign the correct events even when a significant number of false positive events is produced by the system. Furthermore, this result is achieved basically independent of the variance. The mapping to the linguistic representation and assignment performed by FEvA is effectively capable to mitigate the increasing variance. The reference system in comparison gets confused much faster. Increasing the number of false positives events leads

to a decline in correct event assignment rate from about $cear = 0.78$ to less than $cear = 0.4$.

When we look at the corresponding flow completion rates for the same experimental setup depicted in Figure 9.20, we discover a surprising result. Instead of a declining trend in the number of successfully completed flows, the opposite is true. The flow completion rate increases between three and 17 percentage points between $\alpha = 0.0$ and $\alpha = 1.0$ for all variances. For the lower variance value $v = 0.2$ where events are recognized more 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 are complete. But this effect has quite a simple explanation. When the number of false positives increases, the chances for FEvA to assign a wrong event increases, too. Especially when conflicts occur during event the assignment FEvA may map a false positive event to early and falsely. Using this wrong events eventually, leads to the completion of more flows as before. This is an undesired side effect caused by FEvA. But for values of $\alpha \leq 0.2$ this happens for less than 3% of the flows and even fewer for the low noise CMS. In conclusion, FEvA proves itself capable to deal with false positive events, as long as the CMS does not produce very high numbers of false positive events.

Next, we investigate the results for FEvA when confronted with out-of-order events as depicted in Figure 9.21. FEvA achieves consistently high values ($cear \geq 0.94$) even for an increasing amount of out-of-order events. Almost all events are assigned to the correct activities. The use of the event container in combination with the delayed assignment and early mapping to prepared activities mitigate the possible problems. This is a significant improvement on the reference system, where the number of correctly assigned events declines slowly with an increasing number of out-of-order context events.

The achieved flow completion rates are depicted in Figure 9.22. There, we observe the expected drop due to the increasing variance, because the assigned events are not processed any further. We also see a slight decline of about ten percentage points for each variance value, while we increase the number of out-of-order context events.

Considering the missed events depicted in Figure 9.23, FEvA is still able to assign the remaining events accurately, with $caer \geq 0.94$. This is not a surprising result as less events are not a problem for the event assignment algorithm. The reference system has much more trouble to assign events in this situation as no means are taken to replace the missing events somehow.

However a missing event has a very strong impact on the flow completion rate as can be seen in Figure 9.24. While a low number of missing events is somewhat

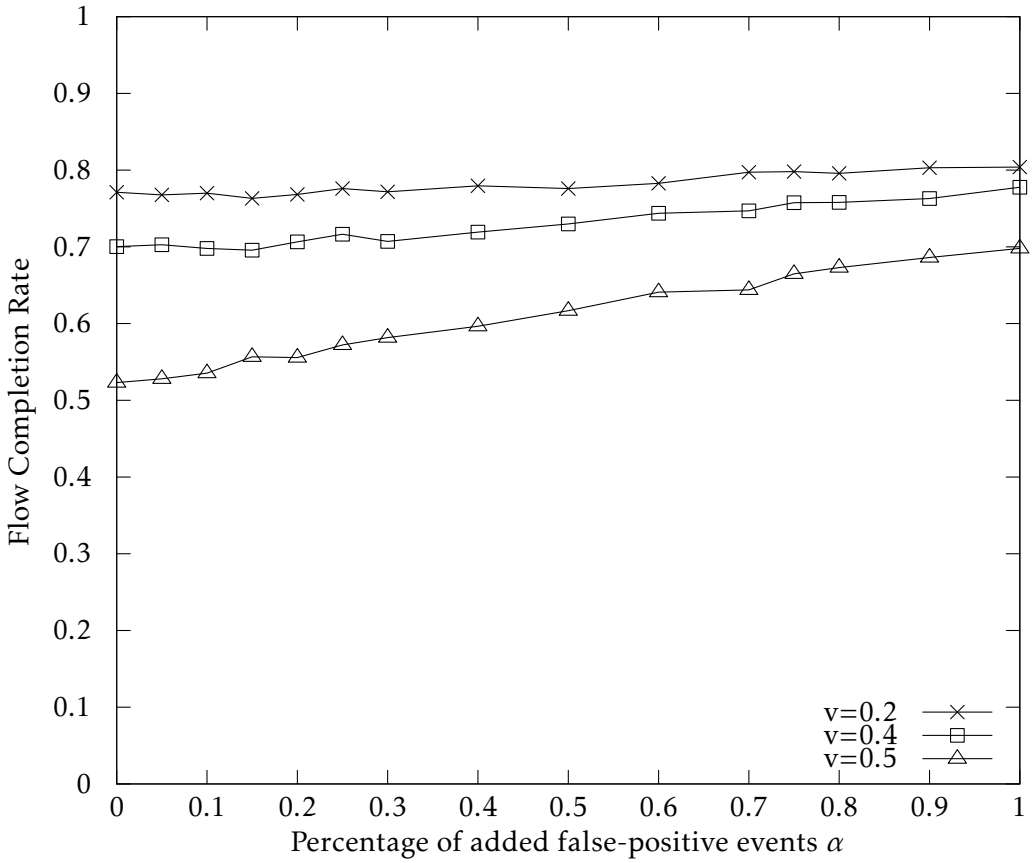

Figure 9.20.: Flow Completion Rate of FEvA for false positive context events

tolerable, the amount of correctly completed flows drops rapidly to a $fcr = 0.07$ when more than a quarter of the events is missing.

Finally, we compare the flow completion rates of FEvA with the reference system for all three system parameters (α, γ, δ). The result is shown in Figure 9.25. The first two bars in each group refer to the false positive events. The reference system gets confused by the false positive events and only achieves rates between $fcr = 0.17$ and $fcr = 0.15$ on average. FEvA is much more stable with respect to the false positives and able to complete between $fcr = 0.52$ and $fcr = 0.76$ of the flows. However the already discussed problem of assigning some of the false positive events to some real activities gives this result a bias.

The middle two bars in each group refer to the out-of-order events. Compared to the other two failure types the naive reference system can deal quite effectively with out of order events achieving flow completion rates of 26% to 28%. But with FEvA the system performs much better witch completion rates between 50% and 74%.

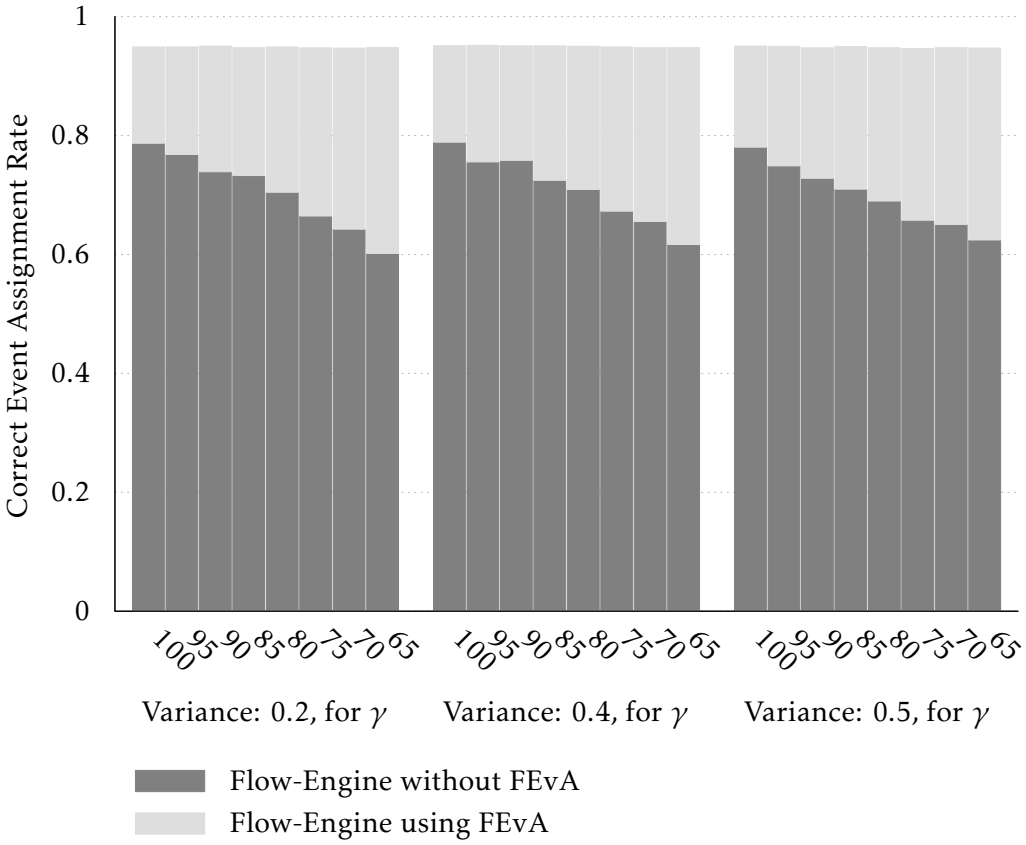

Figure 9.21.: Correct Event Assignment Rate comparison of reference system and FEvA for out of order context events

The last two bars in each group show the values for deleted events. As we already have seen, the flow completion rate using FEvA drops quickly when more than a few percent of the events are missing. This is still much better when compared against the flow completion rate of the reference system under this conditions. For just a few missing events FEvA performs up to 40 percentage points better. However, there is still a lot of room for improvement.

Discussion FEvA provides us with a very mixed set of results. While the correct event assignment rates are surprisingly good when facing errors caused by out-of-order context information and deleted context events, FEvA is somewhat prone to assign false positive events to activities. The inspection of the false positives and their weighting by the activities would have suggested a more robust result. With respect to the flow completion rates achieved, FEvA also provides mediocre results. While for false positive events we suffer from their wrong mapping, for the deleted events FEvA only provides a medium improvement. However the measures taken to allow for a missing event are quite extensive and any further

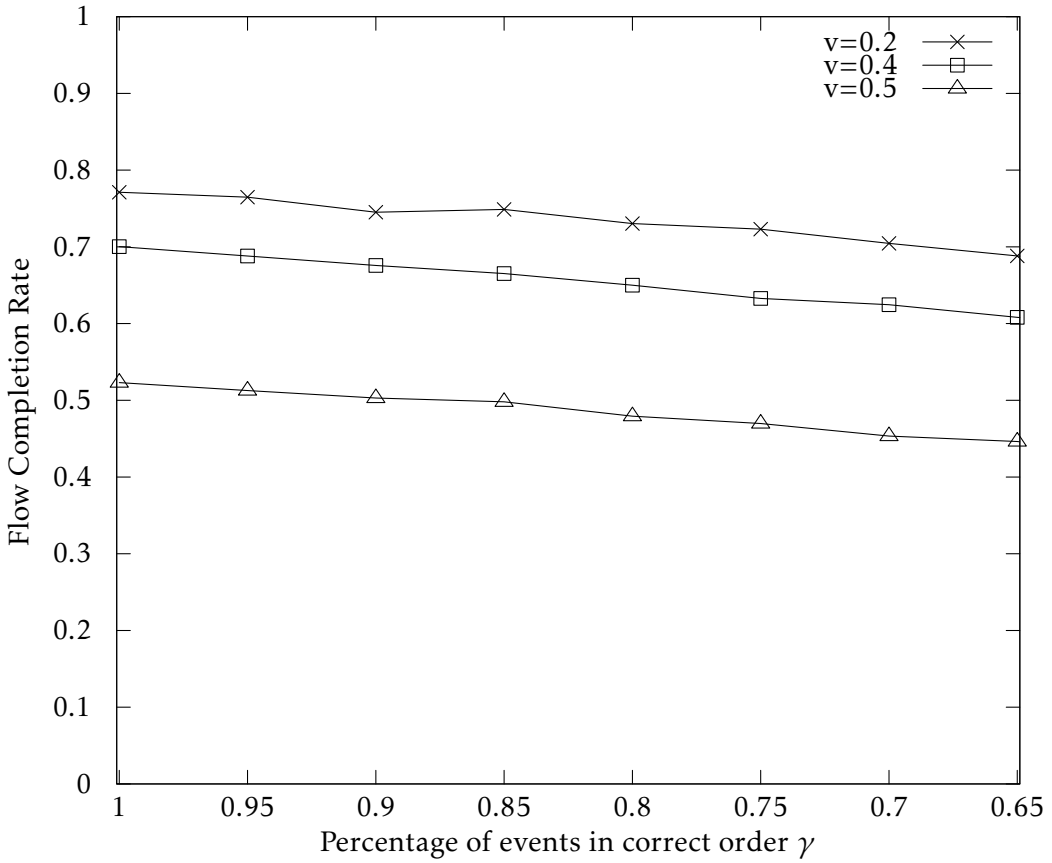

Figure 9.22.: Flow Completion Rate of FEvA for out-of-order context events

relaxation would make the flow execution too self contained, neglecting the actual input.

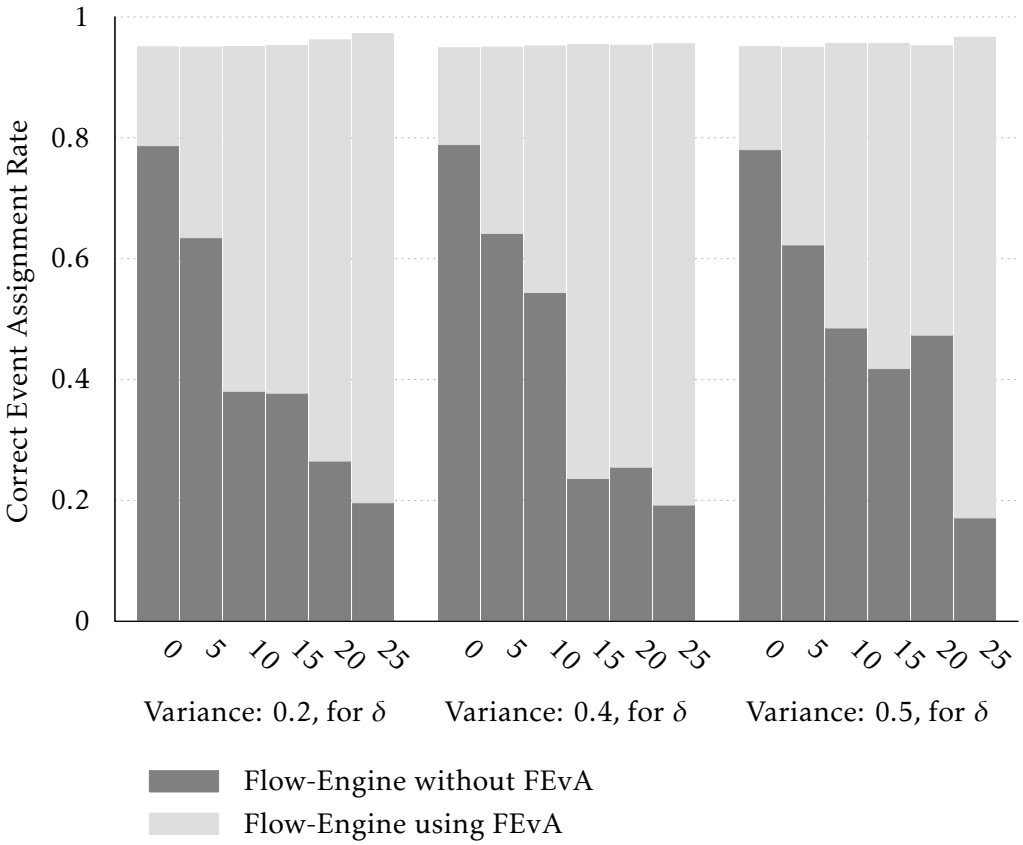

Figure 9.23.: Correct Event Assignment Rate comparison of reference system and FEvA for deleted/missing context events

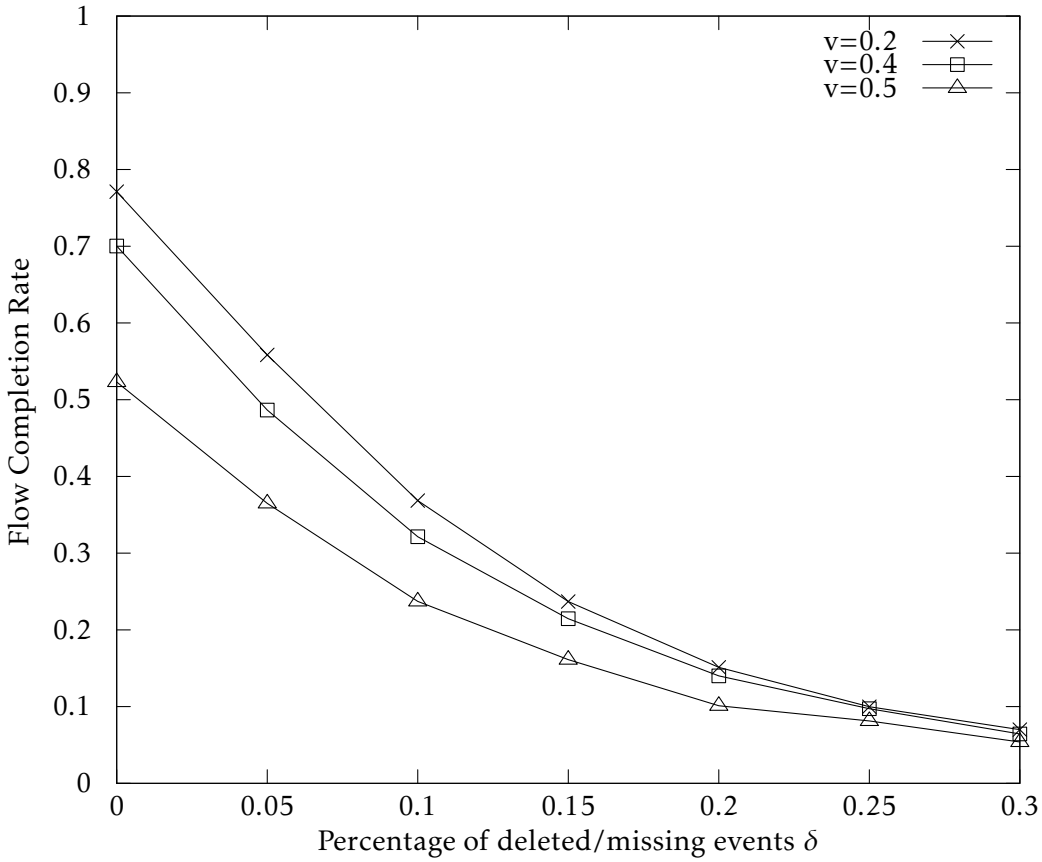

Figure 9.24.: Flow Completion Rate of FEvA for deleted context events

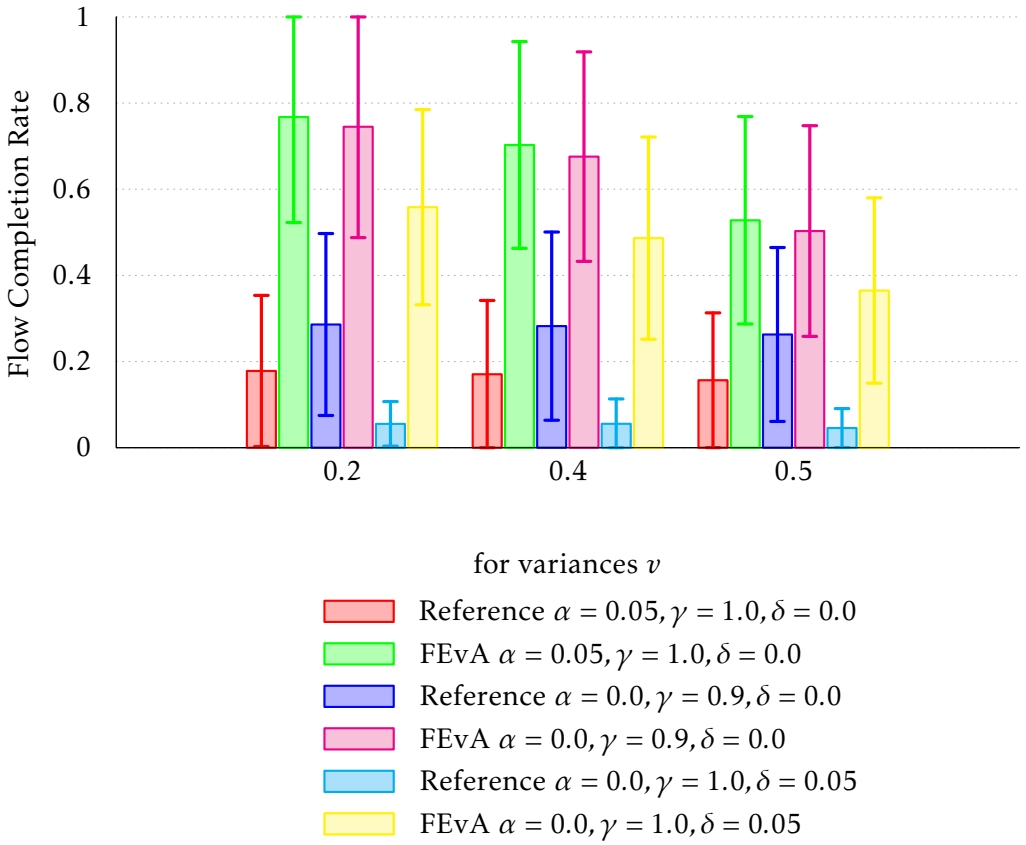

Figure 9.25.: Flow Completion Rate comparison of reference system and FEvA for individual sequence errors

Part IV.

Conclusions and Outlook

10. Conclusions

Modern process-based applications rely on the sophisticated and well established workflow programming model. They are heavily influenced by the wide availability of mobile computing devices and mobile sensors. The applications are driven by context information and human users have to interact with workflows using in various situations using all kinds of different modalities. While the user is supported in his activities the process provides guidance and further can prepare and adapt resources to the users needs. To make this interaction as seamless and unobtrusive as possible a robust detection of the users actions is a key requirement.

In this thesis we have explored the interaction space between human users, subject to activity recognition, and context-driven process-oriented applications. The main goal was to investigate in which ways structural information of a workflow-based application can be used to reduce uncertainty and ambiguity of context information that drive the workflow itself. We have constructed a sound system model that provides us with the basic assumptions required for the system to work properly. In order to apply workflow modeling to the target application scenarios, we investigated current approaches with respect to their modeling flexibility and ways to integrate context information. As a result of this investigation we created the HFM that provides modeling flexibility from both major workflow modeling paradigms imperative and declarative and also is naturally designed to be driven by context information.

Based on the system model we introduced a overall system architecture – FlowPal– that builds upon a standard process management environment. We integrated a CMS and researched a number of methods that can be added as plugins to the architecture and aim to resolve the problems of uncertainty and ambiguity.

The most important plugin is ⋆Con that resolves interpretation issues for received context events. First, we developed a variant, FlowCon, that is suitable for classic imperative workflows only but demonstrates the feasibility of the idea behind ⋆Con very well. We use flow knowledge or more specifically information about the activities in the flow, their structural ordering and historical information as well as information from the currently executed instance, to build a BN that provides us with a second source of information to reason on the probability of the current context information. Having applied this approach with FlowCon successfully to

imperative flows, we constructed FlexCon that uses the same principle but is applicable to the more flexible human-centric flows which could be defined by the HFM. As our evaluation results show FlowCon is capable to increase the accuracy of context events of up to 50% and very reasonable flow completion rates depending on the input characteristics of the CMS. FlexCon is also capable to improve the accuracy of context events of up to 73% which is again very good result. But FlexCon cannot consistently provide these values due to the increased flexibility of the flows. Therefore, the flow completion rates stay behind those for FlowCon. Please note again that a higher flow completion rate means less need for a user to manually intervene. We also tackled the representation and reasoning of uncertain context information in this work and the results have been subsumed in the CE plugin of FlowPal. The investigated variants to represent uncertainty in the system have also a major impact and we found that the overall system could largely benefit from the usage of SL and 4V compared to PT.

Further we developed FEvA, a plugin that tackles issues in the sequence of events that a flow may have to deal with. The goal has been to mitigate the sequence errors and allow the flow to adapt with little user intervention. FEvA fulfills these requirements very well when faced with false positive events or events that arrive the at the application out-of-order, but still has space for improvements on missing events.

A key factor that makes the results we presented resilient is the two phase approach we used to evaluate our algorithms. The case study we conducted in an geriatric nursing home, covers a wide range of requirements relevant for our target application domains, providing us with the necessary insights to setup the large scale simulations to test our algorithms against.

11. Outlook

The FlowPal approach has been so far limited in scope, to keep the complexity especially of the experiments reasonable at first. But there are some natural places to extend them. First, we focused on a single flow application only, neglecting the possible ecosystem of concurrently running flow applications instantiated from different flow models. By extending the structures learned by ⋆Con to all flows in a relevant workplace/scenario a better benefit might be achieved. Another opportunity can be found in the users. So far we assumed that one BN captures the habit of each and every person executing the flow instance. A personalized BN for each user, possibly also spanning multiple flow models, might again yield an overall better result as the habit of this user can be better reflected by the specialized model.

A very interesting next step would be to test the system in a small real-world deployment similar to the scenario we described in Chapter 8. As of now this opportunity was not available. Besides the obvious result of a real world deployment, this would further provide opportunities to come up with suitable meta-heuristics to adapt FlowPal to the actual deployment. Some of the values we found in our evaluations, such as the minimal event ignorance value for the SL CE and the values for the fuzzy membership functions in FEvA might have different optimal values in an actual real world scenario.

Bibliography

[AAD+07] Agrawal, Ashish ; Amend, Mike ; Das, Manoj ; Ford, Mark ; Keller, Chris ; Kloppmann, Matthias ; König, Dieter ; Leymann, Frank ; Müller, Ralf ; Pfau, Gerhard ; Plösser, Karsten ; Rangaswamy, Ravi ; Rickayzen, Alan ; Rowley, Michael ; Schmidt, Patrick ; Trickovic, Ivana ; Yiu, Alex ; Zeller, Matthias: Web Services Human Task (WS-HumanTask), Version 1.0 / Oasis. 2007. – Forschungsbericht

[AAH+09] In: Aalst, W. M. ; Adams, M. ; Hofstede, A. H. ; Pesic, M. ; Schonenberg, H.: *Flexibility as a Service*. Berlin, Heidelberg : Springer-Verlag, 2009. – ISBN 978–3–642–04204–1, 319–333

[AH03] Aalst, W.M.P. van d. ; Hofstede, A. H. M. T.: YAWL: Yet Another Workflow Language. In: *Information Systems* 30 (2003), S. 245–275

[AHEA06] Adams, Michael ; Hofstede, ArthurH.M. ; Edmond, David ; Aalst, WilM.P.: Worklets: A Service-Oriented Implementation of Dynamic Flexibility in Workflows. Version: 2006. http://dx.doi.org/10.1007/11914853_18. In: Meersman, Robert (Hrsg.) ; Tari, Zahir (Hrsg.): *On the Move to Meaningful Internet Systems 2006: CoopIS, DOA, GADA, and ODBASE* Bd. 4275. Springer Berlin Heidelberg, 2006. – ISBN 978–3–540–48287–1, 291-308

[AHH94] Aalst, Wil M. d. ; Hee, K.M. van ; Houben, G.J.: Modelling and analysing workflow using a Petri-net based approach. In: *Proc. 2nd Workshop on Computer-Supported Cooperative Work Petri nets and related formalisms*, 1994, S. pp 31–50

[AKG+10] Altakouri, B. ; Kortuem, G. ; Grunerbl, A. ; Kunze, K. ; Lukowicz, P.: 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. – ISSN 1550–4816

[AMSW09] Aalst, Wil M. ; Mooij, Arjan J. ; Stahl, Christian ; Wolf, Karsten: Formal Methods for Web Services. Version: 2009. http://dx.doi.org/10.1007/978-3-642-01918-0_2. Berlin, Heidelberg : Springer-Verlag, 2009. – ISBN 978–3–642–01917–3, Kapitel Service Interaction: Patterns, Formalization, and Analysis, 42–88

[APS09] Aalst, W.M.P. van d. ; Pesic, M. ; Schonenberg, H.: Declarative workflows: Balancing between flexibility and support. In: *Computer Science-Research and Development* 23 (2009), Nr. 2, 99–113. http://www.springerlink.com/content/f33657685138787m/

[ATM03] Adam, O. ; Thomas, O. ; Martin, G.: Fuzzy Workflows—Enhancing Workflow Management with Vagueness. In: *EURO/INFORMS Istanbul 2003 Joint International Meeting*, 2003, 6–10

[ATV05] Adam, Otmar ; Thomas, Oliver ; Vanderhaeghen, Dominik: Fuzzy-Set-Based Modeling of Business Process Cases. In: *ICCBR Workshops*, 2005, S. 251–260

[AWG05] Aalst, Wil M. P. d. ; Weske, Mathias ; Grünbauer, Dolf: Case handling: a new paradigm for business process support. In: *Data Knowl. Eng.* 53 (2005), Mai, Nr. 2, 129–162. http://dx.doi.org/10.1016/j.datak.2004.07.003. – DOI 10.1016/j.datak.2004.07.003. – ISSN 0169–023X

[Bar81] Barnett, Jeffrey A.: Computational methods for a mathematical theory of evidence. In: *IJCAI'81: Proceedings of the 7th international joint conference on Artificial intelligence*. San Francisco, CA, USA : Morgan Kaufmann Publishers Inc., 1981, 868–875

[BBA05] Barger, T.S. ; Brown, D.E. ; Alwan, M.: Health-status monitoring through analysis of behavioral patterns. In: *Systems, Man and Cybernetics, Part A: Systems and Humans, IEEE Transactions on* 35 (2005), Nr. 1, 22 - 27. http://dx.doi.org/10.1109/TSMCA.2004.838474. – DOI 10.1109/TSMCA.2004.838474. – ISSN 1083–4427

[BDR07] Baldauf, Matthias ; Dustdar, Schahram ; Rosenberg, Florian: A survey on context-aware systems. In: *International Journal of Ad Hoc and Ubiquitous Computing* 2 (2007), Nr. 4, 263–277. http://inderscience.metapress.com/index/1184787H28163T15.pdf

[BFH+09] Bouckaert, Remco R. ; Frank, Eibe ; Hall, Mark ; Kirkby, Richard ; Reutemann, Peter ; Seewald, Alex ; Scuse, David: *Weka manual (3.7.1)*, Juni 2009. http://prdownloads.sourceforge.net/weka/WekaManual-3-7-1.pdf?download

[BG09] Buffett, Scott ; Geng, Liqiang: Bayesian Classification of Events for Task Labeling Using Workflow Models. In: *Business Process Management Workshops*. Milano, Italy, September 2009, 97-108

[BH08] Barker, Adam ; Hemert, Jano: Scientific Workflow: A Survey and Research Directions. In: Wyrzykowski, Roman (Hrsg.) ; Dongarra, Jack (Hrsg.) ; Karczewski, Konrad (Hrsg.) ; Wasniewski, Jerzy (Hrsg.):

Parallel Processing and Applied Mathematics Bd. 4967. Springer Berlin Heidelberg, 2008. – ISBN 978–3–540–68105–2, S. 746–753

[BJ77] Belnap Jr., Nuel D.: A useful four-valued logic. In: *Modern uses of multiple-valued logic*. Springer, 1977, S. 5–37

[BKL10] Bahle, Gernot ; Kunze, Kai ; Lukowicz, Paul: On the use of magnetic field disturbances as features for activity recognition with on body sensors. In: *Proceedings of the 5th European conference on Smart sensing and context*. Berlin, Heidelberg : Springer-Verlag, 2010 (EuroSSC'10). – ISBN 3–642–16981–3, 978–3–642–16981–6, 71–81

[BKS03] Buchholz, Thomas ; Küpper, Axel ; Schiffers, Michael: Quality of Context: What It Is And Why We Need It. In: *In Proceedings of the 10th Workshop of the OpenView University Association: OVUA'03*, 2003

[BTJ+10] Biswas, Jit ; Tolstikov, Andrei ; Jayachandran, Maniyeri ; Fook, Victor Foo S. ; Wai, Aung Aung P. ; Phua, Clifton ; Huang, Weimin ; Shue, Louis ; Gopalakrishnan, Kavitha ; Lee, Jer-En: Health and wellness monitoring through wearable and ambient sensors: exemplars from home-based care of elderly with mild dementia. In: *Annales des Télécommunications* 65 (2010), Nr. 9-10, S. 505–521. http://dx.doi.org/http://dx.doi.org/10.1007/s12243-010-0176-0. – DOI http://dx.doi.org/10.1007/s12243–010–0176–0

[CITR08] Chiao, Carolina ; Iochpe, Cirano ; Thom, Lucinéia H. ; Reichert, Manfred: Verifying Existence, Completeness and Sequences of Semantic Process Patterns in Real Workflow Processes. In: *Proc. of the Simpósio Brasileiro de Sistemas de Informação. Rio de Janeiro: UNIRIO*. Brazil, 2008, p. 164-175.

[CK10] Curbera, Francisco ; Khalaf, Rania: Implementing BPEL4WS: The Architecture of a BPEL4WS Implementation. / IBM T.J. Watson Research Center. 2010. – Forschungsbericht

[CL04] Chakraborty, Dipanjan ; Lei, Hui: Pervasive Enablement of Business Processes. In: *PerCom*, 2004, S. 87–100

[DA99] Dey, Anind K. ; Abowd, Gregory D.: Towards a better understanding of context and context-awareness. In: *In HUC '99: Proceedings of the 1st international symposium on Handheld and Ubiquitous Computing*, Springer-Verlag, 1999, S. 304–307

[DMV+05] Dongen, B. van ; Medeiros, A. de ; Verbeek, H. ; Weijters, A. ; Aalst, W. van d.: The ProM Framework: A New Era in Process Mining Tool Support. Version: 2005. http://dx.doi.org/10.1007/11494744-25.

In: Ciardo, Gianfranco (Hrsg.) ; Darondeau, Philippe (Hrsg.): *Applications and Theory of Petri Nets 2005* Bd. 3536. Springer Berlin / Heidelberg, 2005. – ISBN 978–3–540–26301–2, 1105-1116

[DRK00] Dadam, Peter ; Reichert, Manfred ; Kuhn, Klaus: Clinical Workflows - The Killer Application for Process-oriented Information Systems? In: *Proc. 4th Int'l Conference on Business Information Systems*, Springer, April 2000, 36-59

[DS05] Dustdar, Schahram ; Schreiner, Wolfgang: A Survey on Web Services Composition. In: *INTERNATIONAL JOURNAL ON WEB AND GRID SERVICES* 1 (2005), Nr. 1, S. 1–30

[EFH+09] Eberle, Hanna ; Föll, Stefan ; Herrmann, Klaus ; Leymann, Frank ; Marconi, Annapaola ; Unger, Tobias ; Wolf, Hannes: Enforcement from the Inside: Improving Quality of Business in Process Management. In: *2009 IEEE International Conference on Web Services (ICWS 2009)*. Los Angeles : IEEE Computer Society, Juli 2009

[FFHR11] Fischer, Daniel ; Föll, Stefan ; Herrmann, Klaus ; Rothermel, Kurt: Energy-efficient workflow distribution. In: *Proceedings of the 5th International Conference on Communication System Software and Middleware*. New York, NY, USA : ACM, 2011 (COMSWARE '11). – ISBN 978–1–4503–0560–0, 2:1–2:8

[GH01] Giannakopoulou, Dimitra ; Havelund, Klaus: Automata-Based Verification of Temporal Properties on Running Programs. In: *In Proceedings, International Conference on Automated Software Engineering (ASE'01)*, IEEE Computer Society, 2001, 412–416

[Gin88] Ginsberg, Matthew: Multivalued Logics: A Uniform Approach to Inference in Artificial Intelligence. In: *Computational Intelligence* 4 (1988), S. 265–316

[Gin90] Ginsberg, Matthew: Bilattices and Modal Operators. In: *Journal of Logic and Computation* 1 (1990), S. 1–41

[Gre02] Grefen, Paul: Transactional Workflows or Workflow Transactions? Version: 2002. http://dx.doi.org/10.1007/3-540-46146-9_7. In: Hameurlain, Abdelkader (Hrsg.) ; Cicchetti, Rosine (Hrsg.) ; TraunmÃ¼ller, Roland (Hrsg.): *Database and Expert Systems Applications* Bd. 2453. Springer Berlin Heidelberg, 2002. – ISBN 978–3–540–44126–7, 60-69

[HC03] Hwang, San-Yih ; Chen, Ya-Fan: Personal Workflows: Modeling and Management. In: *Lecture Notes in Computer Science* Bd. 2574/2003, Springer, 2003 (LNCS), S. 141–152

[HCC05] Han, Joohyun ; Cho, Yongyun ; Choi, Jaeyoung: Context-Aware Workflow Language Based on Web Services for Ubiquitous Computing. In: *ICCSA (2)*, 2005, S. 1008–1017

[HCKC06] Han, Joohyun ; Cho, Yongyun ; Kim, Eunhoe ; Choi, Jaeyoung: A Ubiquitous Workflow Service Framework. In: *ICCSA (4)*, 2006, S. 30–39

[HFF+11] Hiesinger, Christian ; Fischer, Daniel ; Föll, Stefan ; Klaus, Hermann ; Rothermel, Kurt: Minimizing Human Interaction Time in Workflows. In: *Proceedings of the Sixth International Conference on Internet and Web Applications and Services (ICIW 2011)*. St. Maarten, the Netherlands Antilles : IARIA, März 2011, 22–28

[HGR07] Hackmann, Gregory ; Gill, Christopher ; Roman, Gruia-Catalin: Extending BPEL for Interoperable Pervasive Computing. In: *IEEE International Conference on Pervasive Services*. Istanbul, 15-20 July 2007, 204 - 213

[HHGR06] Hackmann, Gregory ; Haitjema, Mart ; Gill, Christopher D. ; Roman, Gruia-Catalin: Sliver: A BPEL Workflow Process Execution Engine for Mobile Devices. In: *Proceedings of 4th International Conference on Service Oriented Computing (ICSOC* Bd. 4294, Springer, 2006 (LNCS), 503-508

[HR06] Henricksen, Karen ; Robinson, Ricky: A survey of middleware for sensor networks: state-of-the-art and future directions. In: *MidSens '06: Proceedings of the international workshop on Middleware for sensor networks*. New York, NY, USA : ACM, 2006. – ISBN 1–59593–424–3, 60–65

[HRKD08] Herrmann, Klaus ; Rothermel, Kurt ; Kortuem, Gerd ; Dulay, Naranker: 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* IEEE Computer Society, 2008, 108–113

[IKM+10] Ings, Luc Daveand C. Daveand Clément ; König, Dieter ; Mehta, Vinkesh ; Müller, Ralf ; Rangaswamy, Ravi ; Rowley, Michael ; Trickovic, Ivana: WS-BPEL Extension for People (BPEL4People) Specification Version 1.1 / Oasis. 2010. – Forschungsbericht

[Jøs97] Jøsang, Audun: Artificial reasoning with subjective logic. In: *In 2nd Australian Workshop on Commonsense Reasoning*, 1997

[Jøs11] Jøsang, Audun: *Subjective Logic*. `http://folk.uio.no/josang/publications.html`. Version: September 2011. – Draft book

[KBNL11] Kunze, Kai ; Bahle, Gernot ; Neuburger, Josef ; Lukowicz, Paul: D.2.3 - Final report on the integration of context recognition / Embedded Systems Lab, University of Passau. 2011. – Forschungsbericht

[Kja07] Kjaer, Kristian E.: A survey of context-aware middleware. In: *SE'07: Proceedings of the 25th conference on IASTED International Multi-Conference*. Anaheim, CA, USA : ACTA Press, August 2007, 148–155

[KKR08] Koch, Gerald G. ; Koldehofe, Boris ; Rothermel, Kurt: Higher confidence in event correlation using uncertainty restrictions. In: *28th IEEE International Conference on Distributed Computing Systems Workshops (ICDCSW'08); 2008*, 2008. – ISSN 1545–0678, 417–422

[KL08] Kunze, Kai ; Lukowicz, Paul: Dealing with sensor displacement in motion-based onbody activity recognition systems. In: *Proceedings of the 10th international conference on Ubiquitous computing*. New York, NY, USA : ACM, 2008 (UbiComp '08). – ISBN 978–1–60558–136–1, 20–29

[KWK+09] Kunze, Kai ; Wagner, Florian ; Kartal, Ersun ; Morales Kluge, Ernesto ; Lukowicz, Paul: Does Context Matter ? - A Quantitative Evaluation in a Real World Maintenance Scenario. In: *Proceedings of the 7th International Conference on Pervasive Computing*. Berlin, Heidelberg : Springer-Verlag, 2009 (Pervasive '09). – ISBN 978–3–642–01515–1, 372–389

[LGB05] Lippe, Sonia ; Greiner, Ulrike ; Barros, Alistair: A survey on state of the art to facilitate modelling of cross-organisational business processes. In: *XML4BPM* 1 (2005), S. 7–22

[LR00] Leymann, Frank ; Roller, Dieter: *Production workflow: concepts and techniques*. Prentice Hall PTR, 2000 http://www.amazon.de/Production-Work-Flow-Concepts-Techniques/dp/0130217530

[LSPG06] Lu, Ruopeng ; Sadiq, Shazia ; Padmanabhan, Vineet ; Governatori, Guido: Using a temporal constraint network for business process execution. Version: 2006. http://portal.acm.org/citation.cfm?id=1151736.1151753. In: *Proceedings of the 17th Australasian Database Conference - Volume 49*. Darlinghurst, Australia : Australian Computer Society, Inc., 2006 (ADC '06). – ISBN 1–920682–31–7, 157–166

[LUW10] Leymann, Frank ; Unger, Tobias ; Wagner, Sebastian: On designing a people-oriented constraint-based workflow language. In: *ZEUS*, 2010, 25-31

[LVOS09] Lohmann, Niels ; Verbeek, Eric ; Ouyang, Chun ; Stahl, Christian: Comparing and evaluating Petri net semantics for BPEL. In: *International Journal of Business Process Integration and Management* 4 (2009), Nr. 1, S. 60–73

[LWG+09] Lange, Ralph ; Weinschrott, Harald ; Geiger, Lars ; Blessing, Andre ; Dürr, Frank ; Rothermel, Kurt ; Schütze, Hinrich: On a Generic Uncertainty Model for Position Information. In: Rothermel, Kurt (Hrsg.) ; Fritsch, Dieter (Hrsg.) ; Blochinger, Wolfgang (Hrsg.) ; Dürr, Frank (Hrsg.): *First Internationa Workshop on Quality of Context, QuaCon 2009*. Stuttgart : Springer, June 2009 (LNCS 5786), S. 76–87

[Mur02] Murphy, Kevin P.: *Dynamic Bayesian Networks: Representation, Inference and Learning*, UNIVERSITY OF CALIFORNIA, BERKELEY, Diss., 2002

[NAPI+03] Najafi, B. ; Aminian, K. ; Paraschiv-Ionescu, A. ; Loew, F. ; Bula, C.J. ; Robert, P.: Ambulatory system for human motion analysis using a kinematic sensor: monitoring of daily physical activity in the elderly. In: *Biomedical Engineering, IEEE Transactions on* 50 (2003), Nr. 6, S. 711 –723. http://dx.doi.org/10.1109/TBME.2003.812189. – DOI 10.1109/TBME.2003.812189. – ISSN 0018–9294

[NPP02] Ng, Brenda ; Peshkin, Leonid ; Pfeffer, Avi: Factored Particles for Scalable Monitoring. In: *In Proceedings of the Eighteenth Conference on Uncertainty in Artificial Intelligence*, Morgan Kaufmann, 2002, S. 370–377

[Ped94] Pedrycz, Witold: Why triangular membership functions? In: *Fuzzy Sets and Systems* 64 (1994), 21 - 30. http://dx.doi.org/DOI:10.1016/0165-0114(94)90003-5. – DOI DOI: 10.1016/0165–0114(94)90003–5. – ISSN 0165–0114

[PG94] Pedrycz, Witold ; Gomide, Fernando: A generalized fuzzy Petri net model. In: *IEEE Transactions on Fuzzy Systems* 2 (1994), November, Nr. 4, S. 295 –301. http://dx.doi.org/10.1109/91.324809. – DOI 10.1109/91.324809. – ISSN 1063–6706

[Pro] ProM Fuzzy Miner: *http://www.processmining.org/online/fuzzyminer*, http://www.processmining.org/online/fuzzyminer

[PSA07] Pesic, Maja ; Schonenberg, Helen ; Aalst, Wil M. d.: DECLARE: Full Support for Loosely-Structured Processes. In: *Enterprise Distributed Object Computing Conference, IEEE International* 0 (2007), S. 287. http://dx.doi.org/http://doi.ieeecomputersociety.org/10.1109/EDOC.2007.14. – DOI

http://doi.ieeecomputersociety.org/10.1109/EDOC.2007.14. – ISSN 1541–7719

[PSSA07] Pesic, M. ; Schonenberg, MH ; Sidorova, N. ; Aalst, WMP van d.: Constraint-based workflow models: Change made easy. In: *On the Move to Meaningful Internet Systems 2007: CoopIS, DOA, ODBASE, GADA, and IS* Bd. 4803. Heidelberg : Springer, 2007 (LNCS), 77-94

[RAHE05] Russell, Nick ; Aalst, WilM.P. ; Hofstede, ArthurH.M. ; Edmond, David: Workflow Resource Patterns: Identification, Representation and Tool Support. Version: 2005. http://dx.doi.org/10.1007/11431855_16. In: *Advanced Information Systems Engineering* Bd. 3520. Springer Berlin Heidelberg, 2005. – ISBN 978–3–540–26095–0, 216-232

[RCMR01] Raposo, A.B. ; Coelho, A.L.V. ; Magalhaes, L.P. ; Ricarte, I.L.M.: Using fuzzy Petri nets to coordinate collaborative activities. In: *IFSA World Congress and 20th NAFIPS International Conference, 2001. Joint 9th* Bd. 3, 2001, 1494 -1499 vol.3

[RDD+03] Rothermel, Kurt ; Dudkowski, Dominique ; Dürr, Frank ; Bauer, Martin ; Becker, Christian: Ubiquitous Computing - More than Computing Anytime Anyplace? In: *Proceedings of the 49. Photogrammetrische Woche*. Stuttgart : ifp, September 2003

[Rei10] Reisig, W.: *Petrinetze*. Vieweg-Teubner Verlag, 2010

[RHAM06] Russell, N. ; Hofstede, A.H.M. ter ; Aalst, W.M.P. van d. ; Mulyar, N.: Workflow Control-Flow Patterns: A Revised View / BPM Center Report. 2006. – Forschungsbericht

[RHEA05] Russell, Nick ; Hofstede, Arthur H. M. ; Edmond, David ; Aalst, Wil M. P. d.: Workflow data patterns: identification, representation and tool support. In: *Proceedings of the 24th international conference on Conceptual Modeling*. Berlin, Heidelberg : Springer-Verlag, 2005 (ER'05). – ISBN 3–540–29389–2, 978–3–540–29389–7, 353–368

[RN02] Russell, Stuart J. ; Norvig, Peter: *Artificial Intelligence: A Modern Approach*. 2nd Edition. Prentice Hall, 2002 http://www.amazon.com/Artificial-Intelligence-Modern-Approach-2nd/dp/0137903952/ref=sr_1_2?ie=UTF8&s=books&qid=1272898384&sr=8-2

[Sha92] Shafer, Glenn: Response to the discussion of belief functions. In: *International Journal of Approximate Reasoning* 6 (1992), Nr. 3, S. 445–480

[SLP04] Strang, Thomas ; Linnhoff-Popien, Claudia: A Context Modeling Survey. In: *Workshop on Advanced Context Modelling, Reasoning and Management, UbiComp 2004 - The Sixth International Conference on Ubiquitous Computing, Nottingham/England*, 2004

[TRC+08] Thom, Lucinéia H. ; Reichert, Manfred ; Chiao, Carolina M. ; Iochpe, Cirano ; Hess, Guillermo N.: Inventing Less, Reusing More, and Adding Intelligence to Business Process Modeling. In: *DEXA*, 2008, S. 837–850

[TRCI08] Thom, Lucinéia H. ; Reichert, Manfred ; Chiao, Carolina M. ; Iochpe, Cirano: Applying Activity Patterns for Developing an Intelligent Process Modeling Tool. In: *ICEIS (3-1)*, 2008, S. 112–119

[TRI09] Thom, Lucineia H. ; Reichert, Manfred ; Iochpe, Cirano: Activity patterns in process-aware information systems: basic concepts and empirical evidence. In: *International Journal of Business Process Integration and Management* 4 (2009), Januar, Nr. 2, 93–110. http://www.metapress.com/content/H9X5114788107226

[UBR06] Urbanski, Stephan ; Becker, Christian ; Rothermel, Kurt: Sentient Processes - Process-based Applications in Pervasive Computing. In: *PerCom Workshops*, 2006, S. 608–611

[UELW10] Unger, T. ; Eberle, H. ; Leymann, F. ; Wagner, S.: An event-model for constraint-based person-centric flows. In: *Progress in Informatics and Computing (PIC), 2010 IEEE International Conference on* Bd. 2, 2010, S. 927 –932

[UHW+09] Urbanski, Stephan. ; Huber, Eduard. ; Wieland, Matthias. ; Leymann, Frank. ; Nicklas, Daniela.: PerFlows for the computers of the 21st century. In: *Pervasive Computing and Communications, 2009. PerCom 2009. IEEE International Conference on*, 2009, 1 -6

[Wei91] Weiser, Mark: The Computer for the 21st Century. In: *Scientific American* 265 (1991), September, Nr. 3, 94–104. http://sandbox.xerox.com/want/papers/ubi-sciam-sep91.pdf. – The PDF-file is a reprint.

[WHP11] Wolf, Hannes ; Herrmann, Klaus ; Palauro, Jonas: Fuzzy Event Assignment for Robust Context-Aware Workflows. In: *Proceedings of the Fourth International Conference on Dependability (DEPEND 2011)*, 2011

[WHR09] Wolf, Hannes ; Herrmann, Klaus ; Rothermel, Kurt: Modeling Dynamic Context Awareness for Situated Workflows. In: R. Meersman, P. H. (Hrsg.) ; (Eds.), T. D. (Hrsg.): *OTM 2009 Workshops* Bd. 5872. Vilamoura : Springer-Verlag Berlin Heidelberg, November 2009 (LNCS), 98-107

[WHR10] Wolf, Hannes ; Herrmann, Klaus ; Rothermel, Kurt: 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

[WHR11] Wolf, Hannes ; Herrmann, Klaus ; Rothermel, Kurt: FlexCon – Robust Context Handling in Human-Oriented Pervasive Flows. Version: 2011. http://dx.doi.org/10.1007/978-3-642-25109-2_16. In: Meersman, Robert (Hrsg.) ; Dillon, Tharam (Hrsg.) ; Herrero, Pilar (Hrsg.) ; Kumar, Akhil (Hrsg.) ; Reichert, Manfred (Hrsg.) ; Qing, Li (Hrsg.) ; Ooi, Beng-Chin (Hrsg.) ; Damiani, Ernesto (Hrsg.) ; Schmidt, Douglas (Hrsg.) ; White, Jules (Hrsg.) ; Hauswirth, Manfred (Hrsg.) ; Hitzler, Pascal (Hrsg.) ; Mohania, Mukesh (Hrsg.): *On the Move to Meaningful Internet Systems: OTM 2011* Bd. 7044. Springer Berlin / Heidelberg, 2011. – ISBN 978–3–642–25108–5, 236-255

[WHR13] Wolf, Hannes ; Herrmann, Klaus ; Rothermel, Kurt: Dealing with Uncertainty: Robust Workflow Navigation in the Healthcare Domain. In: *ACM Trans. Intell. Syst. Technol.* 4 (2013), 65:1–65:23. http://dx.doi.org/10.1145/2508037.2508046. – DOI 10.1145/2508037.2508046. – ISSN 2157–6904

[Wik] http://en.wikipedia.org/wiki/File:Accuracy_and_precision.svg

[WKL+09] Wieland, Matthias ; Käppeler, Uwe-Philipp ; Levi, Paul ; Leymann, Frank ; Nicklas, Daniela: Towards Integration of Uncertain Sensor Data into Context-aware Workflows. In: Informatics (LNI), GI-Edition Lecture N. (Hrsg.): *Tagungsband INFORMATIK 2009 – Im Focus das Leben, 39. Jahrestagung der Gesellschaft für Informatik e.V. (GI)*. Lübeck : Lecture Notes in Informatics (LNI), September 2009

[WKN08] Wieland, Matthias ; Kaczmarczyk, Peter ; Nicklas, Daniela: Context Integration for Smart Workflows. In: *Proceedings of the Sixth Annual IEEE International Conference on Pervasive Computing and Communications*. Hong Kong : IEEE computer society, March 2008. – ISBN 0–7695–3113–X, 239–242

[WKNL07] Wieland, Matthias ; Kopp, Oliver ; Nicklas, Daniela ; Leymann, Frank: Towards Context-Aware Workflows. In: Pernici, Barbara (Hrsg.) ; Gulla, Jon A. (Hrsg.): *CAiSE´07 Proceedings of the Workshops and Doctoral Consortium Vol.2, Trondheim, Norway, June 11-15th, 2007*, Tapir Acasemic Press, Juni 2007. – ISBN 978–82–519–2246–3

[YAW13] *YAWL: Credit Card Example*. http://www.yawlfoundation.org/pages/resources/creditcardexample.html. Version: 2013

[Zad65] Zadeh, LA: Fuzzy Sets. In: *Information and Control* 8 (1965), Nr. 3, 338–353. http://linkinghub.elsevier.com/retrieve/pii/S001999586590241X

[Zad86] Zadeh, Lotfi A.: A simple view of the Dempster-Shafer theory of evidence and its implication for the rule of combination. In: *AI magazine* 7 (1986), Nr. 2, S. 85

All URLs have been checked at June 27, 2014.

Curriculum Vitae

Hannes Wolf	
Date and place of birth:	May 17th, 1984; Cottbus, Germany
Nationality:	German
09/2014 – today	Software Engineer for Embedded Devices Bosch Connected Devices and Solutions GmbH, Reutlingen, Germany
04/2012 – 08/2014	Engineer for final test of MEMS-sensors Automotive Electronics, Robert Bosch GmbH, Reutlingen, Germany
03/2008 – 02/2012	Research staff member at the Institute of Parallel and Distributed Systems (IPVS), Universität Stuttgart, Germany
10/2003 – 02/2008	Studies in Computer Science at Universität Stuttgart, Germany Degree: Diplom-Informatiker (Dipl.–Inf.)
06/2006 – 02/2007	ERASMUS exchange studies at University of Limerick, Ireland
07/2002 – 06/2003	National Military Service at "2./233. Gebrigsjägerbataillon" in Mittenwald, Germany
09/1998 – 06/2002	Secondary School and university-entrance diploma at Goetheschule in Ilmenau, Germany