Efficient and Reliable Data Stream Management

Inauguraldissertation

zur
Erlangung der Würde eines Doktors der Philosophie
vorgelegt der
Philosophisch-Naturwissenschaftlichen Fakultät
der Universität Basel

von

Gert Brettlecker
aus Innsbruck, Österreich

Basel, 2008

Genehmigt von der Philosophisch-Naturwissenschaftlichen Fakultät
auf Antrag von:

Prof. Dr. Heiko Schuldt Prof. Dr. Paul Lukowicz

Basel, den 20.05.2008

Prof. Dr. Hans-Peter Hauri
Dekan

Kurzfassung

Aktuelle Entwicklungen in Sensortechnik, eingebetteten Systemen und der Fortschritt in Ubiquitären Informationstechnologien ermöglichen und unterstützen neuartige Anwendungen im Bereich der kontinuierlichen Sensordatenverarbeitung. Anwendungen wie Telemonitoring im Gesundheitswesen oder Verkehrsdatenmanagement verlangen nach einer verlässlichen Verarbeitung von kontinuierlichen Datenströmen, auch Datenstrommanagement (DSM) genannt. Die Anwendungsumgebung ist ausgesprochen fehleranfällig da sie mobile und eingebettete Geräte mit drahtlosen Verbindungen beinhaltet. Um unsere Überlegungen zu motivieren und präzisieren, präsentieren wir eine Beispielanwendung aus dem Bereich Telemonitoring im Gesundheitswesen im Detail. Diese Anwendungen benötigen Unterstützung durch das Kernthema dieser Doktorarbeit; effizientes und zuverlässiges Datenstrommanagement. Natürlich werden Effektivität und Flexibilität als notwendige Voraussetzungen für erfolgreiche Anwendungen nicht ausser Acht gelassen.

Das Hauptaugenmerk der Arbeit umfasst drei Bereiche: Erstens analog zu den Isolationsebenen in SQL definieren wir ein Modell für zuverlässiges DSM basierend auf Zuverlässigkeitsebenen. Dazu beschreiben wir notwendige Konsistenzbedingungen in verteiltem DSM. Zweitens präsentieren und analysieren wir einen neuartigen Algorithmus für zuverlässiges, verteiltes DSM, namentlich Effizientes Koordiniertes Operator Checkpointing (ECOC). ECOC basiert auf dem präsentierten DSM Model und garantiert sowohl verlustlose als auch verzögerungsbegrenzte Zuverlässigkeitsebenen. Daher ist ECOC in kritischen Anwendungsbereichen einsetzbar welche keinen Verlust von Daten erlauben. Der ECOC Ansatz erlaubt feingranulare Sicherungen auf Operatorebene welche eine effiziente und flexible Nutzung der vorhandenen Ressourcen im Netzwerk ermöglichen. Im Weiteren ist ECOC optimiert um die Kosten für Zuverlässigkeit in Bezug auf Netzwerk-, CPU-, und Speicherbedarf zu begrenzen. ECOC unterstützt komplexe Ausführungsgraphen von Datenstromoperatoren mit Vereinigungen, Verzweigungen und sogar Zyklen. Drittens präsentieren wir eine detailliere Analyse des Laufzeitverhaltens des ECOC Algorithmus im fehlerfreien und fehlerbehafteten Fall in einem Servernetzwerk und einem Netzwerk von mobilen Geräten.

Zum Schluss unserer Arbeit demonstrieren wir die Anwendbarkeit unseres Ansatzes mit Hilfe einer anschaulichen Implementierung einer Telemonitoring Beispielanwendung aus dem Gesundheitswesen unter Verwen-

dung von verschiedenen physiologischen und anderen Sensoren. Die Analyse des Laufzeitverhaltens und der Demo-Prototyp basieren auf der OSIRIS-SE Implementierung eines verteilten zuverlässigen Datenstrommanagementsystems. Die Implementierung in Java ermöglicht die Ausführung der gleichen Software sowohl auf mobilen als auch auf Server-Geräten.

Abstract

The proliferation of sensor technology, especially in the context of embedded systems, and the progress of ubiquitous computing strongly supports new types of applications that make use of streams of continuously generated sensor data. Applications like telemonitoring in healthcare or roadside traffic management systems urgently require reliable data stream management (DSM) in a failure-prone distributed setting including resource-limited mobile and embedded devices. In order to motivate and illustrate our considerations, we investigate an application in the field of telemonitoring for e-health in detail. Telemonitoring applications in healthcare are demanding the key issue of this thesis, namely *efficient and reliable data stream management*. Due to its importance for applicability, effectiveness and flexibility is also considered in this work.

The main contribution of this thesis is threefold. First, in analogy to the SQL isolation levels, we define a model for reliable DSM based on levels of reliability and describe necessary consistency constraints for distributed DSM. Second, we present and analyze a novel algorithm for reliable distributed DSM, namely efficient coordinated operator checkpointing (ECOC) based on this model. We show that ECOC provides lossless and delay-limited reliable data stream management and thus can be used in critical application domains such as healthcare, where the loss of data stream elements cannot be tolerated. The ECOC approach considers fine-grained backups at operator level, which allows for the flexible and efficient usage of available resources in a network. Moreover, ECOC is optimized to reduce the overhead of checkpointing and to support complex stream process execution graphs, which include joins, splits and even cycles within data stream flows. Third, we present detailed performance evaluations of the ECOC algorithm running in a network of both stationary server nodes and mobile, resource-limited devices.

Finally, the applicability of our approach is presented by an e-Health telemonitoring demo prototype developed with real-world sensors within this thesis. All evaluations and the demo application are based on the distributed DSM infrastructure prototype OSIRIS-SE. The Java implementation allows for running the same software on both mobile and stationary devices.

Acknowledgements

Firstly, I want to thank my advisor, Prof. Dr. Heiko Schuldt, for valuable discussions, advice, and encouragement during the years of my Ph.D. studies. The work presented in my thesis has started at the University of Health Sciences, Medical Informatics and Technology (UMIT) in Tyrol, Austria. In the year 2006 the work migrated to University of Basel, Switzerland. Prof. Dr. Heiko Schuldt gave my the opportunity to join his group. He was already supporting my work when we were together at UMIT although I was not member of his research group.

Secondly, I want to thank my former adviser at UMIT, Prof. Dr. Hans-Jörg Schek for giving me the opportunity to start a Ph.D. career and supporting me with valuable discussion and advices in the beginning of this work.

I wish to thank my second reviewer, Dr. Paul Lukowicz at University of Passau, Germany, for his willingness to review my thesis and for his time and effort in doing this.

This thesis could not have been possible without financial support from different projects. I am graceful that the following projects have supported work done within this thesis:

- *Health Monitoring* (IT-based support and care for people in need of care), funded by Health Information Technologies Tyrol (HITT) and Tiroler Zukunftsstiftung at UMIT.

- *DELOS* (Network of Excellence on Digital Libraries), funded by the European Union in the 6th Framework Programme at UMIT.

- *DELOS* (Network of Excellence on Digital Libaries), funded by the Swiss State Secretariat for Education and Research (SER) under contract No. SBF 03.0546-3 at University of Basel.

I am very grateful to all of my colleagues and friends in the Department of Computer Science at University of Basel and at UMIT, for many helpful discussions and a pleasant working environment. In particular, I thank Michael Springmann for many valuable discussions on the topic and help on revising the final version of the thesis.

Basel, May 2008

Gert Brettlecker

Contents

List of Figures

List of Tables

1

Introduction

1.1 Trends and Applications

In recent years, the proliferation of pervasive computing, wireless communication and sensor technology has spawned a variety of new applications in the area of *Data Stream Management* (DSM). In general, these applications are continuously monitoring the real world environment to extract and derive relevant information from multiple sensor streams. In the near future, ubiquitous and pervasive computing are starting to infiltrate people's daily life and generate an increasing amount of continuously generated data [Mat01]. In particular, research in the field of *sensor networks* is focusing on necessary sensor node hardware, wireless networking, and basic processing algorithms. For example, *TinyOS* [HSW+00a] provides an open-source operating system for wireless sensor networks.

A challenging task is the extraction of relevant information coming from heterogeneous data streams produced by various different devices and sensors. In particular, we have to consider the inherent distributed setting containing multiple different components, i.e., mobile and embedded devices, where failures are likely to happen. A conference on research challenges in information systems [J+03] has highlighted the task to build "systems you can count on" as one of five major research issues in IT for the future. These new circumstances in pervasive and ubiquitous computing environments raise new issues in information management which traditional *database systems (DBMS)* never designed to meet [GO03]. Similarly, conventional workflow and process support systems are tailored to execute processes at dedicated points in time and not to keep them continuously running to process streams of data. A proposed solution for the task of managing continuously gen-

erated data in near real-time is *data stream management (DSM)*. Various groups [BBC+04, C+03, BBD+02, SKK04, YG03, CDTW00] are currently in research for new paradigms and techniques to handle and process information flows like a DBMS does with static data. In addition to those rather generic approaches investigated by these groups, we focus in this thesis on an application-driven approach. Therefore, an important part of this thesis is the presentation of major application areas for DSM particularly in healthcare and the analysis of their general properties. Extensibility and flexibility of the proposed DSM infrastructure allows for executing disease – or more general – scenario specific DSM operations. While DSM is quite promising, it is not enough. The increased number of heterogenous components, (mobile) devices, and platforms leads to an increased failure probability. Reliability and provable correctness are new challenges [J+03] that are of utmost importance for various real-world applications, i.e., in healthcare.

1.1.1 Trends and Applications in Healthcare

Within this thesis we emphasize in particular on an important field for applications of data stream management — applications in healthcare. The following, we present three reasons why we focus on these applications:

- Demand for automated DSM in future to increase quality of patient's life.
- Challenging environment incorporates mobile and stationary devices.
- Healthcare applications require highest reliability. Failures may have severe consequences.

Telemonitoring in healthcare (TM) allows healthcare institutions to continuously monitor their patients at home while they are out of hospital in an online fashion, which is especially useful for measuring the effects of treatments under real-life conditions as well as for managing various chronic diseases or even to immediately react to critical pathological changes. Technically, TM aims at dynamically gathering, managing, processing, and storing physiological data usually provided by a network of smart sensors.

The demand for TM is increasing due to the progression of chronic ailments in an aging society. Chronic ailments such as cardiovascular diseases, hypertension, and diabetes affect a significant number of the western population [Ame07, Sop05]. According to a survey of the American Heart Association [Ame07], in particular cardiovascular diseases are the leading cause of death in the US. Europe is facing a similar situation [Sop05]. Moreover,

if we consider our aging society [Eur05], the amount of elderly people suffering from one or more chronic diseases will be increasing. Chronic ailments are frequently combined with general age-related impairments (e.g., visual and tactile impairments) and mental diseases (e.g., Alzheimer). The healthcare system has to face this problem and increase its effectiveness in order to avoid exploding costs.

Online telemonitoring is a rather novel domain and still topic of intensive research [SB06, Sac02, WSN+00, AWL+04, Mob03]. Research in this fields needs interdisciplinary cooperation of experts from various fields such as electrical engineering, computer science, networking, information management, (biomedical) signal processing, and - of course - medicine. In the field of electrical engineering, telemonitoring greatly benefits from recent trends in smart sensors and ubiquitous and wearable computing (e.g., smart shirts [GTW03, Sen07], ring sensors [ASR+03], or smart bandages [NAS00]). As a result, a new generation of sensor systems currently emerges that allow for non-invasive monitoring of an increasing number of patients and diseases. In the field of networking, wireless network connections with both minimal stand-by and transmission power consumption are needed to increase the lifetime of sensors and to avoid unnecessary perturbation of the system. Experts from biomedical signal processing provide efficient and effective algorithms for the (pre-) filtering of sensor signals and the processing of these signals that allows to extract critical and/or medical relevant information. Moreover, experts from the medical domain are needed to elaborate use cases and evaluate the benefit of TM for the therapy of patients. Finally, from an information systems point of view, an efficient and reliable software infrastructure is needed that supports continuous processing of data streams, like the system and implementation described in this thesis. Until today, reliability of DSM infrastructures has received little attention in research given it's utmost importance for various applications, i.e., healthcare applications. Apparently, reliability is of utmost importance for healthcare applications. Patients and caregivers need to fully count on the TM system. Failures have to be compensated by the TM system in order to continue TM. Compensation can by done by using alternative available nodes for processing or alternative communication channels. Nevertheless, if the failure situation is too severe to continue monitoring, the TM system has to raise an appropriate alarm in a reliable way. Such an alarm has to inform both the patient and the caregiver that online monitoring is currently not available and give hints on how the issue can be fixed as soon as possible. For example, informing the patient to replace batteries of a mobile device or calling a technical assistance service. The existence of a failure in a TM system without knowledge to the patient or caregiver may have severe consequences.

Recent developments are trying to push the TM scenario even more ahead. *E-inclusion* is emphasized by the EU as a strategic objective [Eur06] in order to allow elderly and impaired people to benefit from modern ICT and prevent the further growth of a so-called "Digital Divide" in society. Research in the context of e-inclusion tackles the ambitious scenario of *ambient assisted living (AAL)*. AAL aims to develop a smart home infrastructure in order to support an independent living for the handicapped people as long as possible at home. The projects in this domain are most challenging, because all issues of TM applications above are relevant to a high degree and simultaneously here. Elderly people tend to suffer from chronic diseases, that require TM. Additional age-related impairments make independent living at home difficult and therefore assistance for daily activities is required, and comfort services will foster social contacts and prevent them from isolation. From the perspective of sensor technology, this does not only require wearable sensors but also sensors that are integrated into the smart-home environment (e.g., "intelligent carpets" [SL07] measuring the position of a person and also his/her activity, i.e., whether a person is active –moving– or whether he/she fell down).

1.1.2 Trends and Applications in other Domains

The work presented in this thesis is not limited to the healthcare domain. Similar issues arise also in other applications domains as for example:

- Road Traffic Management
- Industrial Process Automation
- Environmental Monitoring
- Structural Health Monitoring
- Power Grid Monitoring

Monitoring and managing the traffic on roads is an application area of increasing importance during the last decades [CHKS03]. Obviously, increasing the utilization of streets is far cheaper than building new streets. Recent developments apply sensor networks on the streets (toll monitoring) as well as built into cars (GPS, road conditions). The huge amount of streaming traffic information data needs to be processed in a fast and reliable manner, so that the relevant information is extracted and relayed to all participants of road traffic in a soft real-time fashion. In near future, the number of sensors and also the number of participants receiving information will reach up to the number of millions, with some of the sensors delivering readings every second.

Therefore, scalable, reliable DSM systems are required. These systems can provide various benefits for road traffic management, like improved safety, better utilization of roads, lower traveling times, increased consumer comfort, and —recently most important— reduction of carbon dioxide emissions. Last but not least, reliability of such applications is an important aspect. The breakdown of a traffic management system due to a failure will cause heavy congestions with severe consequences for the environment and the economy.

Another application area with similar data stream management demands is industrial process automation [CHK+06]. Modern factories are equipped with vast amount of sensors monitoring the production processes and providing continuous streams of process and machine data. Extraction of relevant information will provide important support for business decision making or optimization of the production processes. Of course, reliability of the DSM infrastructure is again important in this scenario. Failures may have severe economic consequences or even influence the safety of the running industrial process, i.e., the factory workers.

Environmental monitoring with sensor networks has also demand for reliable data stream management. Considering surveillance systems of volcanic activity [WALJ+06], tsunamis [CLD08], or oil spills [HJR+98], malfunction may have severe impact on people and environment.

Structural health monitoring [CFP+06] assesses the structural integrity of bridges, buildings, and aerospace vehicles. Obviously, the reliability of the monitoring system is of utmost importance to prevent harm due to suddenly occurring degradation in these structures.

Power grid monitoring [YDHH06] is another field of application for DSM. The increasing power consumption of the industrial world leads to increasing utilization of existing power grid lines. Better control of the utilization by an reliable DSM system prevents from economic burden and danger of blackouts due to overload situations.

1.2 Requirements for DSM infrastructures

The presented applications by far exceed the capabilities of existing database systems [GO03] in terms of processing and storing continuous data streams but also in terms of support for distributed pervasive computing environments. Similarly, conventional workflow and process support systems are tailored to execute discrete processes at dedicated points in time. Data stream processing, however, require processes to run continuously. Services as basic activities in discrete processes are now accompanied by *operators* which continuously process data streams in continuously running *stream processes*.

Contrarily to the world of discrete services, data stream operators are *stateful*. This means that these operators produce results not only depending on the current input stream data but also input stream data received previously by continuous modification of an operator state during runtime, e.g., a data stream operator calculating the average heart beat frequency within the last hour. Obviously, in case of failures the recovery of operator states is necessary to produce correct results. Special requirements are also needed for the storage of streams, or for joins between streams in order to combine the data produced by different sensors. The bottom line is that existing information management infrastructures, which only support discrete processes or transactions, have to be extended in order to continuously execute stream processes in a volatile pervasive computing environment. Nevertheless, the presented applications are not limited to streaming data because e.g. telemonitoring as well as road traffic management requires traditional discrete processing of data. For example, results from TM have to be stored in the electronic health record of the patient or the traffic volume at critical crossroads today have to be stored in statistic database for later analysis. Therefore, we propose in this thesis an *integrated information management infrastructure* supporting both traditional discrete process management and data stream management (see Fig. 1.1).

In the following, we present a list of the four most important requirements a DSM infrastructure has to provide:

- Integrated Continuous DSM and Discrete Information Management
- Reliability
- Support for Distributed Pervasive Environments
- Flexibility

Reliability: For various application domains (e.g., health monitoring and e-Inclusion), a stringent requirement is that the underlying information management infrastructure implementing DSM is *highly dependable* since its correct functioning may be potentially life-saving or preventing other severe events. Contrarily, the distributed environment consisting of many connected devices, both mobile and stationary, combined with unreliable wireless communication implies a *high failure probability* compared to distributed computing scenarios involving only administered server computers and Ethernet connections, e.g., it is very likely that a roadside sensor gets damaged due to an accident or even caused by animals or a wireless connection gets temporarily hampered by interference. Applications in these areas have to be fault tolerant because failures are much more likely to occur than in centrally

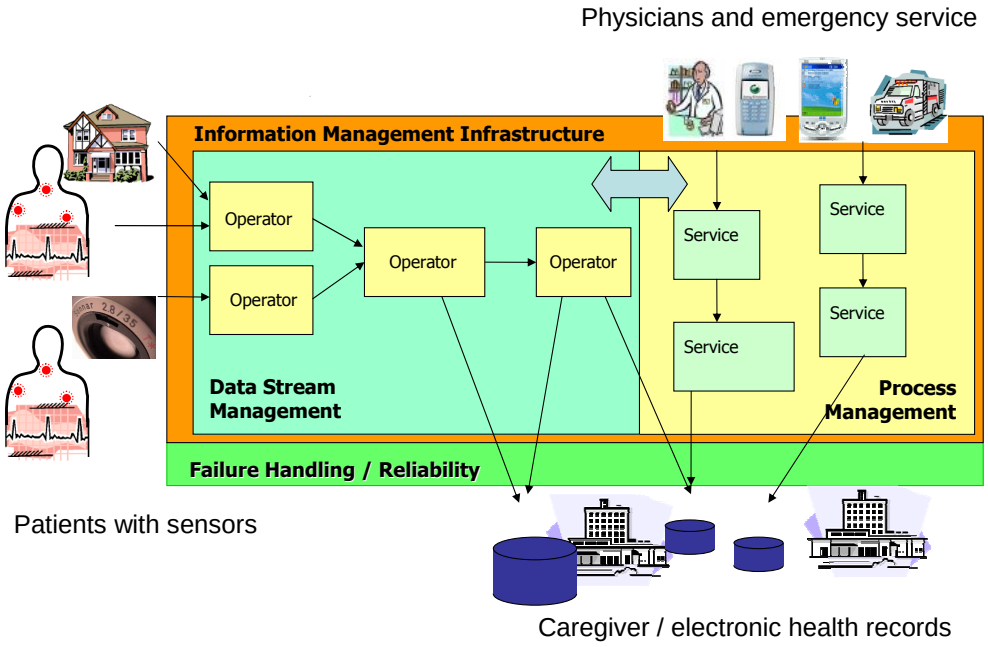


Figure 1.1: Information Management Infrastructure for Healthcare.

controlled applications (e.g., banking server). This means the DSM infrastructure has to cope with failures and therefore has to apply *sophisticated failure handling* techniques in order to compensate and continue the application seamlessly or to invoke reasonable failure handling at application level. In here, dependability comes with several flavors.

First, the information management infrastructure has to be *highly available and reliable* in order to provide dedicated quality of service guarantees, for example in terms of the real-time aspects of the data streams being processed during runtime. Critical situations have to be detected immediately and alarms have to be raised on the spot; no delay due to overload situations of the infrastructure can be tolerated. Even in case of failures, e.g., a mobile device is going down because of empty battery, the stream processing should be continued seamlessly if there are enough available resources (devices) that can backup for the failed ones.

Second, the process-based applications have to be built in a way that correct failure handling is guaranteed (e.g., by following a reliability model as for transactional processes [SABS02]). This also means that applications, since they are vital to their users, have to be verified a design-time whether they will behave correctly, even in failure situations.

In general, different application domains have different requirements in terms of reliability and quality of data processing. Currently, there is no common model for specifying quality and reliability constraints for DSM applications demanded from a DSM infrastructure.

Distributed Environment: As an additional requirement imposed by the presented applications, we have to consider DSM as a task inherently performed in a *distributed pervasive computing environment*. Therefore, a novel DSM infrastructure has also to support *embedded and mobile devices*, like a patient's PDA of the health telemonitoring scenario or a roadside sensor unit of the traffic management scenario. A crucial requirement on this infrastructure for DSM is to take into account that users and patients, i.e., when being monitored in an out-of-hospital environment, are usually mobile. Mobility of users and patients poses a set of challenges to the design and development of an information infrastructure for DSM. However, even when a device (e.g., a PDA carried by the patient) is disconnected or the some device fails, data streams produced by some local (body) sensors still have to be processed and stored by using the still available resources. Thereby it is important to take limited CPU and storage resources into account and therefore use resources *efficiently*.

Flexibility: In order to allow for fast adaptivity and reusability, DSM and discrete processing applications are not monolithic but should be designed out of basic building blocks (e.g., programs for sensor data filtering or processing, noise reduction, access to medical databases or electronic patient records, etc.) as it is state of the art in the world of service oriented architectures [Ley05]. Each of these DSM building blocks (or *data stream operators*) provides certain services that can be invoked. Application development in this context therefore requires to seamlessly combine these building blocks into a coherent whole rather than developing programs from scratch. Workflows or processes are a means to combine existing services also in the world of DSM. For example, the disease pattern of patients varies over time and the system should allow to tailor stream processing to the current monitoring needs of an elderly person or a patient. This includes new types of sensors, new types of services, new types of processes, etc., that must be supported.

1.3 Contribution

In this dissertation, we present a novel information management infrastructure for integrated reliable data stream and discrete process management. A vision for the future is that such an infrastructure is able to cope with the discrete and streaming data processing demands of modern distributed applications mainly coming from the pervasive and ubiquitous computing area so as the presented applications. Moreover, we want to emphasize in this thesis on achieving a very high degree of reliability for these applications although they are running in a volatile distributed setting where failures are very likely to happen. Nevertheless, the infrastructure is able to use available resources efficiently and compensating failures seamlessly and transparently for the application.

In order to realize the vision of this information management infrastructure and meet the requirements for DSM infrastructures stated in Section 1.2 this dissertation makes the following main contributions:

- Overview of healthcare application research and issues targeted by application research (see Chapter 2).

- Definition of a formal model for data stream management which is capable of covering the application issues (see Chapter 4). Moreover a formal reliability model for DSM is covering the important reliability aspects and is giving a formal framework for the reliability strategy algorithms. In order to proof correctness of algorithms, we define different reliability levels of DSM (see Section 5.1) which describe the degree of allowed failures in data streams by still be considered as correct DSM processing. The presented reliability levels cover the three orthogonal failure conditions of data streams; *loss* of data stream elements, *delay* of data stream elements, and *order* of data stream elements. In this thesis, we have intentionally not modeled accuracy of data stream elements as reliability criteria of the general DSM model, because we state that in particular for healthcare applications inaccurate DSM processing is generally not tolerable. Inaccurate data stream elements are considered as invalid and therefore result in loss of data stream elements.

- Development of a suitable efficient reliability strategy for DSM to allow for effective and efficient operator migration in case of failures or overload situations in Peer-to-Peer fashion by keeping lossless reliability. As opposed to to most research in the field of DSM [HBR+05, BBMS05, BBC+04, SHB04, C+03], the presented reliability strategies for DSM in

this thesis are performing at the *data stream operator* level in a distributed environment. Reliability at the operator level and not at the level of a whole stream processing node allows for a fine grained load redistribution in case of failures or overload situations, which is called *operator migration* (see Section 5.7). Based on operator migration, this thesis presents and evaluates a new reliability protocol, called *Efficient and Coordinated Operator Checkpointing* (ECOC) (see Section 5.10), to reduce the drawbacks of passive standby approach presented in [HBR+05], i.e., high runtime and recovery overhead. Optimizations of ECOC support real world stream processing scenarios having complex distributed operator graphs including joins, splits and even cycles.

- Design and implementation of an integrated DSM and process management infrastructure as successor of the existing process management infrastructure OSIRIS [SWSS04, SST+05, SWSS03] (see Chapter 6), which is called *OSIRIS-SE* (Stream Enabled) [BS07, BSS06, BSS05]. Since the original OSIRIS has been implemented in proprietary C++ limited to Windows platforms, we have re-implemented the basic OSIRIS process management functionality in Java in order to allow for platform independence, i.e., to support mobile and embedded device platforms that offer an appropriate Java virtual machine. Moreover, we have integrated the support for *reliable execution* of continuously running DSM processes or *stream processes* in the Java-based OSIRIS-SE infrastructure. Essentially, stream processes have to be continuously fed with incoming sensor data. An important requirement is also to provide an easy-to-use graphical interface that can be used by non-programmers (e.g., physicians or care personnel) to design new or to revise existing patient- and/or disease-specific stream processes. For this reason, OSIRIS's process design tool *O'Grape* [WSN+03] has also been extended to support the design of stream processes.

- The experimental evaluation in Chapter 7 is twofold. One aspect is to provide a demonstrator illustrating a simplified real world telemonitoring scenario by incorporation of a set of real world sensors and mobile devices. The second aspect is performance evaluation on both server hardware and mobile devices. For these evaluations we still use real sensory data which is processed during evaluations. Moreover, these evaluations cover both the performance during the normal (failure-free) runtime of the system and the performance during the phase when recovering from failures.

1.4 Structure of the Thesis

This thesis is organized as presented in Fig. 1.2. Chapter 2 presents an overview of application areas in the healthcare domain of relevance for DSM. This chapter presents a visionary healthcare application scenario used throughout the thesis. In Chapter 3, we describe the DSM infrastructure based on the hyperdatabase concept and how the concept has been extended in order to support DSM processing. In Chapter 4, we introduce a formal model for DSM. The model is based on assumptions that are derived from the analysis of application specific requirements. In Chapter 5, we introduce reliability levels and consistency within a distributed DSM system. Based on this formalism, reliability algorithms are presented which guarantee correct DSM processing even in case of failures. The presentation of the reliability algorithm is based on the formalism presented in Chapter 4. Moreover, this formalism allows to proof the introduced reliability algorithms theoretically. Chapter 6 describes the implementation of the DSM infrastructure in OSIRIS-SE. Technical details on the implementation of the basic OSIRIS system and the DSM extensions of are presented. Chapter 7 empirically proofs the applicability and performance of the presented DSM infrastructure and in particular the presented reliability algorithms through exhaustive evaluations within the real-world infrastructure implementation of OSIRIS-SE. In Chapter 8, we survey related work in the field of reliable data processing and data stream management. Moreover, we discuss the differences compared to our approach. In addition, the chapter introduces relevant application specific research and derives common issues with respect to a DSM infrastructure required by these applications. Finally, Chapter 9 concludes in summarizing the impact of the presented work and discussion of open and future research issues.

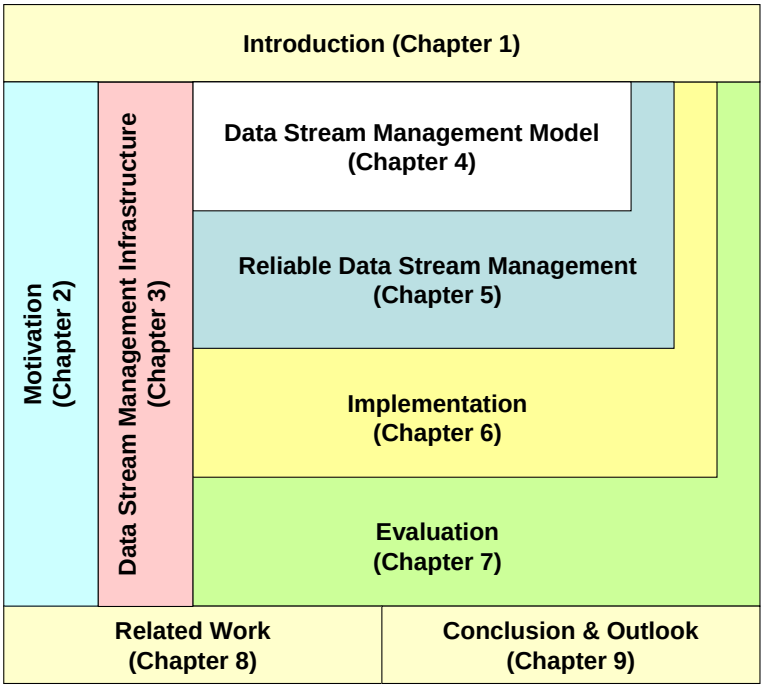


Figure 1.2: Structure of the Thesis.

2

Motivation

In this chapter, we motivate the applicability of DSM to various application domains. In particular, we introduce applications in the e-health field with relevance for DSM and discuss the influences of new trends such wearable and pervasive computing. In order to get a more precise view of the application, we present a motivating telemonitoring application scenario in detail and propose a novel DSM application system as a vision for the future. This example application scenario is used throughout the remainder of the thesis for motivation and illustration purposes.

2.1 e-Health Applications with DSM Demand

2.1.1 Telemonitoring Applications

Chronic ailments such as cardiovascular diseases, hypertension, diabetes, overweight, or cognitive impairments (e.g., Alzheimer disease) are important candidates to develop novel e-Health systems and consequently giving benefit to a significant number of the western population [Ame07, Sop05, Eur05]. In this thesis, we focus on *telemonitoring (TM)* applications in healthcare. TM applications enable healthcare institutions first to take care of and control therapies regarding their patients while they are out of hospital. Secondly, they serve as instrument for performing research and for accomplishing medical studies. Thirdly, they allow for triggering of emergency services in case of severe health conditions. Finally, they can offer additional comfort services as by-product, like assistive services, information services and communication services. As a consequence, the patient's disease will be better managed with less hospitalization (which usually has physical and emotional impact)

and higher quality of life [BBGA03, RBS+02, RSB+01, RKH03, SSP+01]. Additionally, TM applications provide a major financial benefit compared to traditional care [DPSB01]. A prognosis for the year 2013 [HAHK02] expects the use of direct permanent monitoring of patients' vital signs, as well as the direct synchronous transfer of this sensory data to significantly increase. A study in [BBGA03] compares the outcomes of using a TM system for home health care of chronic heart failure patients to traditional home nurse visits. The results show that TM improves the patients health status and reduces cost of care. Pathologic changes in vital signs are detected early [KWSB02], which allows for easy intervention while the patient is still at home and avoids hospital readmission. Hypertension is a second major health risk in our western society. A randomized controlled trial [RBS+02] shows that TM improves the diagnosis of essential hypertension. Diabetes is a third chronic disease with a high prevalence which can benefit from TM. At Columbia University, a TM project was conducted in which 1.500 participants are monitored [SSW+02b]. The outcome of the project is that fast intervention and feedback from the healthcare provider will allow for better control and maintenance of glucose level and blood pressure compared to usual care. Obviously, a vital requirement in telemonitoring is that the application provides a high degree of reliability and availability, since it can potentially be life-saving. Therefore, TM is considered to significantly enhance the quality of patients' lives and to increase overall quality of care, even in out-of-hospital conditions with major reduction of costs [DPSB01].

In order to remotely monitor a patient's health status, a network of wearable sensors is attached to the subject's body. Depending on the necessities of the concrete medical case, the sensors take periodic measurements of physiological parameters such as *electrocardiogram (ECG)*, heart rate or blood pressure as well as activity/context parameters such as location, velocity, or acceleration [Ana03]. Since the utility value of the raw sensory data is rather low [MFHH02], it has to be processed (e.g., filtering, error compensation, feature extraction) and locally stored for buffering and further analysis. These functions can be partially performed by smart sensors, wearable, intermediary devices (e.g., PDA, smart phone) as well as by the patient's surrounding IT-infrastructure (e.g., base station at home, see Fig. 2.1). For system and device control, user interface components have to be accessible, which can also display relevant health and system status information to the patient [WCP+99]. Long-range communication (e.g., Internet over UMTS) allows for transmission of already preprocessed health data of interest to the responsible care provider for observation, diagnosis, and treatment purposes. Furthermore, in case of emergency or critical deviation of the patient's health parameters from the norm, appropriate actions such as an alarm or a call

for paramedics should be taken automatically on behalf of the system's local or remote data analysis and interpretation. As this overview shows, a system consisting of multiple, physically and logically distributed information processing components is necessary in order to remotely manage a person's health status.

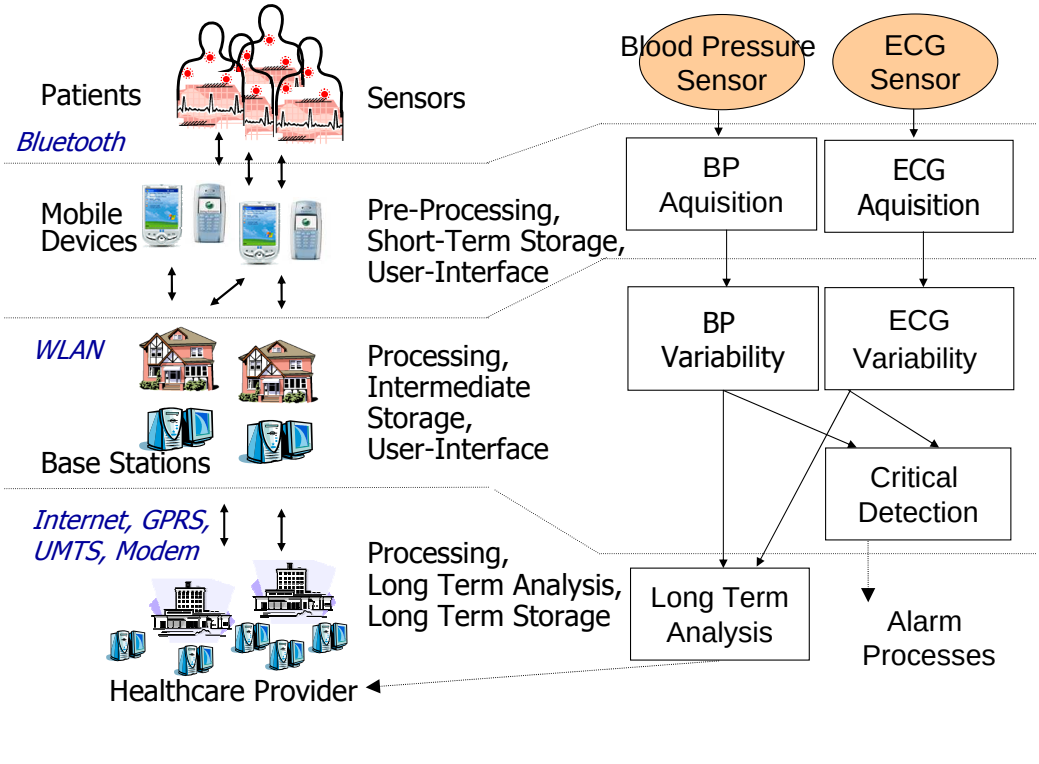


Figure 2.1: Information Processing between Patient and Caregiver in a Telemonitoring Application.

Figure 2.1 illustrates the typical setting of a telemonitoring application. The left hand side displays the hardware and communication needed to process the data streams along from the patient to the caregiver. On the right hand side, we illustrate an example stream processing composed of different specialized *data stream operators* which are executed at different devices within the system. The interconnected operators build up a processing mesh, called *stream process*. More details on stream processes will be presented in Chapter 3.

2.1.2 e-Inclusion & Ambient Assisted Living

Recent developments are trying to push the TM scenario even more ahead. *E-inclusion* is emphasized by the EU as a strategic objective [Eur06] in order to allow elderly and impaired people to benefit from modern ICT and prevent the further growth of a so-called "Digital Divide" in society. Research in the context of e-inclusion tackles the ambitious scenario of *ambient assisted living (AAL)*. AAL aims to develop a smart home infrastructure in order to support an independent living for the handicapped people as long as possible at home. From the perspective of sensor technology, this does not only require wearable sensors but also sensors that are integrated into the smart-home environment (e.g., "intelligent carpets" [SL07] measuring the position of a person and also his/her activity, i.e., whether a person is active –moving– or whether he/she fell down).

In this kind of monitoring applications, it is therefore necessary to consider the *context* of a person. According to [Dey01], context refers to any information that can be used to characterize the situation of an entity. Where an entity can be a relevant person, place, or object. Research in the area of pervasive computing provides techniques and algorithms in order to detect the current context based on sensor information [BKL06, BGL07].

Applied to the area of AAL, a person without current activity may be fine at night when being located in the bedroom, but when the person does not show any activity at daytime when being located in the floor, an emergency situation is very likely. We are considering furthermore people suffering from special age-related impairments and mental diseases like Alzheimer. In this case, even more emphasis has to be put on monitoring the activity of the person together within the current context. A warning has to be generated, for instance, when the person tries to leave the apartment without having switched off the oven. By all these extensions, the number of sensors, data sources, etc., and therefore also the number of data streams that have to be processed will significantly increase compared to the TM scenario. This, of course, has severe consequences on the requirements to the underlying information management infrastructure.

The integration of home automation is an essential aspect of assisted living for elderly or impaired people. The projects in this domain are most challenging, because all issues of TM applications above are relevant to a high degree and simultaneously here. Elderly people tend to suffer from chronic diseases, that require TM. Additional age-related impairments make independent living at home difficult and therefore assistance for daily activities is required, and comfort services will foster social contacts and prevent them from isolation. A project at the University of Florida [GHM02], called *mobile*

Patient Care-giving assistant for Alzheimer (mPCA) is aiming in this direction. The system assists the patient in difficulties of daily life. A location tracking system tracks the position and orientation of the patient and other important objects (e.g., pills, pets). This context information is used to find out which activities the patient intends to do. For instance, if the patient is standing in front of the oven, the system can assume that the patient is going to cook. The system will also remind the patient to perform important tasks (e.g., turning off the oven after cooking, taking the medication). All patient activities are monitored and deliver important information for the physician on the progress of the disease.

2.1.3 Wellness and Lifestyle Monitoring

More and more people today attach great value to a health-conscious lifestyle, which is driven by prevention of diseases and to maintain or increase their health status. In this context intelligent monitoring applications emerge that are using continuous sensor data and wearable devices. These applications are similar to the telemonitoring and ambient assisted living applications, where both, relevant physiological parameters and the context of users are continuously monitored and analyzed to allow for correct interpretation. Nevertheless, the demand for reliability is not as important as in the two previously presented fields of application. Also, the complexity of the overall system is not as big as in the field of ambient assisted living applications. Usually, for wellness monitoring the setup is smaller and more focused to specific tasks. In particular obesity is a primary concern addressed by the users of such systems. For this reason, let us assume the necessary sensors are invisibly in the clothes and other everyday life articles (e.g., clock, armbands [Bod07], etc.) and therefore are fully wearable and ubiquitous. The important aspect in this application area is the unobtrusiveness of the used wearable sensors and devices, like in an unobtrusive armband worn on the back of the upper arm [Bod07]. Since the users are not suffering from diseases, there is a lack of willingness to restrict their every-day behavior due to the monitoring. Similarly to the previous applications, the wearable devices are wirelessly connected to a base station which receives the pre-processed streaming data and extracts information about things such as physical exercise quantity, seating position, meal times, rest periods, etc. and process them in correlation with some continuously gathered physiological health parameter, like skin impedance, skin temperature, heat dissipation. Deviations from normal state and suggestions to improve the lifestyle can be communicated either to the user immediately, or presented later as a result of a long-term analysis. In general, this kind of applications can be also used for early recog-

nition of health changes and may support medical research on development of health problems and on prevention of diseases.

2.2 DSM in Healthcare Present and Future

The following fictitious monitoring scenario will serve as illustration of our statements and examples throughout the course of this thesis:

2.2.1 The current State

Fred, aged 68 and retiree, lives alone in a house of his own. In the EU, 23 million adults are suffering from diabetes [Sop05]. So does Fred, maybe undiagnosed since a long time. High blood pressure and diabetes often occur together and if left untreated can lead to serious consequences for the heart [NHEB01]. Unfortunately due to the long time of untreated diabetes and hypertension, Fred has developed *congestive heart failure* (CHF) 1.5 years ago. CHF is defined as a disorder causing the heart to lose its ability to pump blood efficiently to the rest of the body. CHF may develop over weeks, months or years. Numerous risk factors can compound the effects of CHF. The most controllable of them include smoking, obesity, excessive alcohol intake, high-fat and/or high-sodium diets, hypertension, diabetes, lack of sufficient physical activity and lack of daily consumption of vegetables and fruits. From the patient's perspective, common symptoms associated with CHF include shortness of breath, swelling of the legs and ankles, pulse irregularity and palpitations, and difficulty with eating or sleeping. Clinically, what is typically occurring is fluid retention around the lungs, changes in blood pressure and enlargement of the heart. Due to his age, Fred also shows slight signs of dementia, which unfortunately affects the effectiveness of his personal disease treatment. For example, sometimes Fred forgets to take his medication or does not drink enough. Without an assistive telemonitoring system, Fred has to do manual random sampling of his blood pressure, blood glucose level, heart rate, and body weight. For further examination, he has to consult his family doctor frequently. Nevertheless, this manual treatment does not prevent Fred from regularly hospitalization due to dramatic degradations of his state of health. Recently, a water accumulation in Fred's lungs caused an acute medical emergency. During the hospital stays, Fred's cardiac balance is restored by medicamentous treatment. Unfortunately, this balance is very unstable and hard to maintain by manual random sampling of physiological signs.

2.2.2 Vision for the Future

As a vision for the future, Fred's caregiver will decide to equip him with a wearable health monitoring systems consisting of a smart shirt [GTW03, Sen07, Viv07], a ring sensor [ASR+03], a glucose measuring watch [Ani07], and a PDA for local processing, intermediate storage, and wireless communication. This wearable setup will allow for unobtrusive monitoring of ECG, heart rate, respiratory and sweating rates, blood pressure, blood glucose level, blood oxygen saturation as well as motion activities, sensed with an inbuilt accelerometer. Fred's PDA will wirelessly communicate with the base station of his smart home system in order to extract and forward relevant streaming data to the caregiver. Overall, the distributed setup of the telemonitoring application will be according to Figure 2.1. Besides that, Fred's smart home infrastructure also aggregate additional context measurements. For this reason, Fred's physical activity is detected by acceleration sensors attached at Fred's body and an integrated positioning system in the smart home environment. Additionally, an electronic scale is measuring body weight and fat, an electronic medication dispenser controls medication, and an smart bed sheet is acquiring Fred's physical activity while sleeping. Various electric appliances like, e.g., the oven, fridge, TV set are also connected to the smart home infrastructure to enable assistive and comfort services. This additional measurements allow for detection of context information. In order to interpret Fred's vital signs even more appropriate this additional context information is needed, e.g., ECG signals vary if Fred is running or sleeping and in order to make correct medical interpretation this information is necessary. Additionally, sensors in the toilet are controlling the amount of water Fred is loosing through urination. Keeping the water amount in balance is very important in treatment of CHF. If Fred is using toilets outside of his home he has to roughly estimate this information and enter it in his smart phone manually. Since Fred shows slight dementia these manual tasks are likely forgotten. In order to remind him, a microphone in Fred's smart phone can be use to detect the sound pattern of a flushing toilet or sensors in his underwear can detect whether the underwear is removed. Also Fred's drinking habits have to be logged to control the water balance. A swallowing sensor [AAB99] integrated in Fred's necklace could by used to remind Fred to enter the liquid amounts into his smart phone after eating or drinking.

Feedback and Additional Services

The smart home infrastructure also offers an feedback channel to Fred by using different screens integrated in his environment, e.g., his smart-phone, his TV set, or the monitor of his computer. An important issue regarding the

feedback system is to gather the attention of persons with moderate dementia, like Fred. An attention-capturing application is activated when a particular task needs to be done at a given time, e.g., to go to the medication dispenser and take the medication or doing some regular physical exercise in order to keep healthy. By using positioning information, the smart home infrastructure is able to choose the best suitable display system for the current context to get Fred's attention [GHM02]. Once the attention is captured, the system is able to give feedback information.

For instance during a long-term therapy, some of the monitored measurements are likely subject to changes, i.e., they have a general tendency. As a result, the physician is able to send various instructions to Fred, for instance to tell him to alter the dose rate of a certain medicine, which may be automatically done by the medication dispenser within his smart home, to be more vigilant about the diet or physical activity, or simply tell him that everything is ok. The goal of treatment for Fred is to reduce his cardiac workload and keep his glucose level constant. Accomplishing this needs a multifaceted approach involving patient education, behavior modification, medication and close medical supervision. Giving Fred tools and support to better manage his disease can help him avoid the physical and emotional impact of emergency-department visits and hospitalizations, as well as the stress surrounding the financial burdens related to stationary or nursery care.

Additional comfort and assistive services are also offered by the application system. For example, Fred may forget to turn off the oven after preparing a meal. If the smart home infrastructure detects that Fred is leaving the kitchen and going to sleep (by using positioning information, acceleration sensors, and physiological sensors) the oven is turned off automatically. Another assistive service can display the cooking recipe on a screen in the kitchen. Fred has already chosen the meal in the morning and the system has prepared the shopping list. After checking his food stock and the fridge, Fred has adapted the list on his smart phone and went shopping. Nevertheless, Fred is still able to change his meal when he is in the supermarket. Fred can select another meal on his smart phone which is contacting the smart home environment and an adapted shopping list is transmitted. RFID technology used for food products may even allow the smart home environment to check the food stock for Fred. Also the stock of medicine in the medication dispenser needs to be controlled. If the dispenser is running out of medication, the smart home environment is automatically getting a prescription from his caregiver. The prescription is send to the closed pharmacy and Fred just needs to go there and get the medication. RFID's incorporated in medication packages allow for the smart home environment to check whether Fred is filling the medication dispenser correctly. Another important aspect of Fred's smart home

environment is social networking. The system should improve the social contacts of impaired persons like Fred with other persons in his surroundings, which may also be impaired. Interconnecting smart home environments offer a variety of new beneficial application in order to improve Fred's autonomy and quality of life by supporting neighborly help and reducing feeling of social isolation. For example, Fred is going shopping to a supermarket nearby daily. Sometimes, he accidentally meets his neighbor Anna, enjoys a small talk, and helps her with the shopping bag. Anna lives in Fred's neighborhood, is 76 years old, and suffers from osteoporosis. But Fred would not like to disturb Anna and call her to go shopping together. In order to improve such social contacts, Fred's smart home environment can coordinate his habits with habits of other smart home residents. In a neutral and anonymous way, the smart home infrastructure can ask different persons with common habits to join their activities today, e.g., asking Anna and Fred to go shopping together. Certainly, the smart home infrastructure has to keep the privacy of the residents and common activities need common agreement. Other habits that may be coordinated by smart homes are card playing, doing physical activities like walking or biking, or just meet for coffee and small talk in a nearby café.

The DSM Infrastructure of the Future

The DSM infrastructure is continuously processing the data streams acquired from Fred's physiological sensors and context information coming from the smart home. An important issue in this scenario is the continuous correlation of data from the different sources, which are integrated into the DSM infrastructure. For example, the correlation between skin moisture, ECG, respiration rate, and acceleration information allows for detection whether sweating was caused by physical activity or physiological disturbance. Based on continuously processed data streams the current context and state of Fred's health is derived. This data stream processing is individually defined by Fred's caregiver in a graphical "boxes and arrows" approach by combination of basic building blocks. The relevant medical information from the DSM infrastructure allows the caregiver to control treatment and immediately intervene by e.g., modifying medication through the automatic medication dispenser. Additionally, the caregiver is able to define individual health thresholds with are critical for Fred. If the infrastructure detects such a critical health situation the caregiver is automatically alerted or in severe case the emergency process is activated (e.g., calling the ambulance). Additional obvious emergency cases like collapse, unconsciousness, or cardiac arrest are also detected by the smart home infrastructure and trigger appropriate emergency handling.

Roughly estimated all data acquired about Fred will likely exceed a GByte per day. Therefore a efficient and reliable information management DSM infrastructure is needed, which will provide the services to analyze incoming data streams, and to extract and to forward relevant information to the patient and the care provider in charge. Even considering Fred's care provider, which is monitoring thousands of patients, reliable and scalable data stream processing is a vital requirement for such systems. Furthermore, the infrastructure will provide a flexible platform for different kinds of monitoring applications and allow for monitoring patients suffering on various chronic diseases by supporting individual profiles for each patient. Fred may suffer from additional diseases in the future (and therefore also new types of data sources and processing operations have to be integrated then). Data stream processing results at caregiver side are not only stored and queried later, additionally the information is needed to be routed directly to the relevant recipients, e.g., the physician in charge for Fred to adapt his medication and treatment or the emergency service closest to the Fred home in case of severe patterns in his vital parameters. Moreover, the analysis of aggregated continuously monitored medical information over hundreds to even thousands of patients supported by Fred's caregiver offers new means for medical research.

In case of changes in Fred's health condition, his physician in charge will be automatically informed, and is able to retrieve all medical important data. For this reason, the DSM infrastructure is not only able to perform data stream management, but also offers access to distributed medical health records. Non-streaming tasks, like accessing these health records are based on the orchestration of discrete services. Therefore, the DSM infrastructure supports also process management to orchestrate discrete service calls. On the other hand, results about Fred's health condition derived from telemonitoring are automatically updated in Fred's health record. This example illustrates the need for an integrated DSM and process management infrastructure, where stream processing and discrete processes are working seamlessly together. Without a telemonitoring system, Fred's has to either move into a nursing home with loosing his independence or he has to deal with the risk that degradation of his health condition is not detected. Given this facts, Fred will accept a certain degree of interference due to the telemonitoring system on his daily life.

However, the most important fact is that the infrastructure has to guarantee a certain degree of reliability. Patient's and caregiver's have to count on the system, and failures may have severe consequences. Unfortunately compared to a centrally controlled server system, the distributed setting including embedded and mobile devices is much more vulnerable to failures (e.g., a mobile device runs out of battery or a wireless connection is disturbed). Still

the infrastructure can guarantee a certain degree of reliability by making use of other redundantly available devices in a transparent and effective way. This is based on the assumption that homes and hospitals of the future are pervaded with computers that are participating nodes in this infrastructure. Since also connections between nodes are subject to failures, the infrastructure has to deal with intermitted connectivity. In this case, disconnected parts of the system still have to operate locally based on decentralized control. In such situations, the separated parts can perform data (pre-)processing and simple hazard detection. After reconnection, data stream processing is automatically and smoothly continued and the filled intermediate buffer are worked off in order to provide full fledged medical, activity, and context monitoring again.

Supported with this reliable integrated DSM infrastructure, Fred will feel safe to live independent at home. Dramatic degradations of Fred's health condition or sudden critical events, e.g., falls, will be detected automatically by the system and an emergency service is alarmed. On the other hand, this system supports medical research by continuously analyzing the physiological parameters of thousands of patients and acquiring the outcome of treatment methods.

3

Data Stream Management Infrastructure

We propose an integrated information management infrastructure supporting *reliable* execution of user-defined processes, both workflow and streaming processes as needed for DSM applications. *Stream processes* are composed of *data stream operators*, or shortly operators, (e.g., medical sensors, sensor signal processing tasks, etc.) whereas workflow processes are based on discrete service invocations such as data processing services, data storage services, or event notification services, to mention only a few. Common in both cases is that operators or services, respectively, have to be combined in a given order (*control flow*) and that data has to be passed between them (*data flow*). Moreover, the two are tightly connected to each other since workflow processes can be invoked by DSM results e.g., an operator of a stream process monitoring our patient Fred detects a heart attack. In this case *reliability* becomes a *life-saving* requirement. Also workflow processes can affect stream processes (e.g., Fred's physician is setting up new threshold for continuous blood pressure monitoring within the regular treatment process). In all these cases, an infrastructure is needed that supports processes, that provides dedicated execution guarantees for processes, and that is able to scale with the number of processes and services or operators. For this reason, our infrastructure considerations are based on *Hyperdatabase (HDB)* systems [SBG+00, SSSW02, SSW02a] which already provide a sophisticated infrastructure for workflow processes supporting (Web-)service compositions. Because of the strong relation between stream processes for DSM and processes for workflow management and similar quality requirements regarding reliability, we propose the extension of HDB systems in order to support reliable DSM.

3.1 The Hyperdatabase Vision

The introduction of relational database systems in the 80's has led to a new infrastructure and main platform for development of data-intensive applications. Data independence was considered to be a major breakthrough: programmers were freed from low-level details, e.g., how to access shared data efficiently and correctly, given concurrent access. But by now, the prerequisites for application development have changed dramatically. Usually, specialized application systems or components are well in place. Applications are no longer built from scratch but rather by combining services of existing components into processes or workflows. These components implement their own data management and provide application services to the outside. Hence, similar to supporting access to shared data, current infrastructures need to provide access to shared services.

A *Hyperdatabase (HDB)* [SBG+00, SSSW02, SSW02a] is such an infrastructure that supports the execution of processes on top of distributed components using existing services. The HDB provides transactional guarantees for processes, fault tolerance for process execution, sophisticated routing strategies to dynamically choose among the available components, and allows to balance the overall load among them. Hence, the HDB provides database-like functionality, but not at the data level but at the level of access to services.

The Hyperdatabase system itself does not provide any application functionality but, by combining specialized application services, supports the definition and reliable execution of dedicated processes (this is also known as "programming-in-the-large" [DK76]). When different specialized application services are made available to the HDB-platform, users can define and run powerful application processes by making use of these services. HDB-processes themselves are wrapped by service interfaces again. Therefore, a process can be invoked just like any other service (and used in other processes as well).

A basis of the Hyperdatabase vision is the model of transactional processes [SABS02]. These processes contain two orders on their constituent services: a (partial) *precedence order* specifies regular execution while the *precedence order* is defined for failure handling purposes (i.e., alternative execution paths). Data flow between services within a process can be defined independently of control flow. Activities in a process are invocations of application services. Ideally, the transactional behavior of each application service is known. This transactional behavior includes information on compensation (how can the effects of a service execution be semantically undone; this is needed for compensation purposes in case a failure in a process execution occurs) and on whether a failed service can be re-invoked (retriability). In ad-

dition to transactional execution, the Hyperdatabase concept emphasizes on *reliability* at various levels, from reliable messaging to persistent queues.

Moreover, an important part of the Hyperdatabase is a graphical modeling tool to enable design and validation of processes in a boxes and arrows approach without sophisticated programming skills. Figure 3.1 illustrates a screenshot of an example workflow process definition within the process design tool O'GRAPE [WSN+03]. The example process is taken from our telemonitoring example application (see Section 2.2) where also the execution of workflow processes based on discrete service calls is needed. The process definition illustrates a process allowing Fred to order medication with his PDA. The process consists of three activities. Firstly, Fred enters his medication request on his PDA (green circle symbol). Secondly, Fred's physician is informed and has to sign the electronic prescription (green square symbol). Thirdly, given a signed prescription the pharmacy is now able to deliver the ordered medication to Fred (green cross symbol).

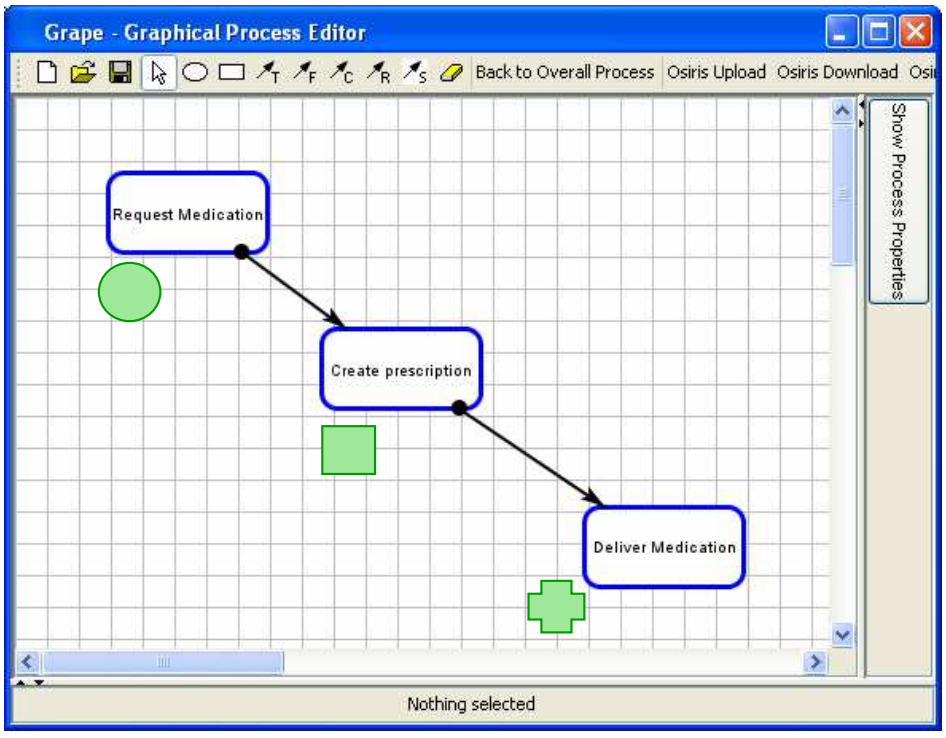


Figure 3.1: An Example Process Definition

3.1.1 The HDB-Architecture

The architecture of a Hyperdatabase is driven by the goal of implementing a true peer-to-peer process execution engine in a reliable and efficient way. Therefore, process execution involves only those machines of the network that offer a service required by the process definition. Because they are the only potential candidates for execution. Conceptually, the HDB system is a *Service-oriented Architecture (SoA)* [Ley05], where service intercommunication is done by passing messages through queues or pipelines which are part of the HDB-middleware. There is no central peer that controls the execution of process instances during process runtime. As a consequence, distributed process execution necessitates efficient metadata management and replication in order to provide enough information locally to allow for a local decision on how a process instance has to be routed. The HDB distinguishes between application-specific services, called *application services*, and services providing general purpose functions needed by all applications, called *system services* (see Fig. 3.2). Examples of Hyperdatabase system services are repository services, replication services, and process execution services, to name just the most important. An example of an application service is the "Deliver Medication" service of the example process (see Fig. 3.1).

While system services are part of the HDB-middleware, application services can be any callable services provided by third parties. The architecture of a Hyperdatabase consists of two parts. Firstly, each peer of the network providing application services runs a small middleware layer, *HDB-layer*, that provides local functionality for process instance navigation and routing (see Fig. 3.2). Secondly, an HDB runs a number of additional system services. One particular type of system services, which are not deployed at all service providers, are *global repositories* for metadata management (see e.g., "Service Repository" in Fig. 3.2). The global repositories hold information about process definitions, subscription lists, service providers, load information, etc. The local HDB-layers that actually execute processes should not have to query meta information from the global repositories during process runtime. Rather a specialized system service, the *replication service* is applying a push mechanism that replicates those parts of meta information towards the local HDB-layers in advance in a lazy way. For this reason, the providers are able to fulfill their tasks based on locally available information. For instance, if a peer is involved in executing a certain process, the definition and any changes to that definition are pushed from the process repository to the local HDB-layer of the peer.

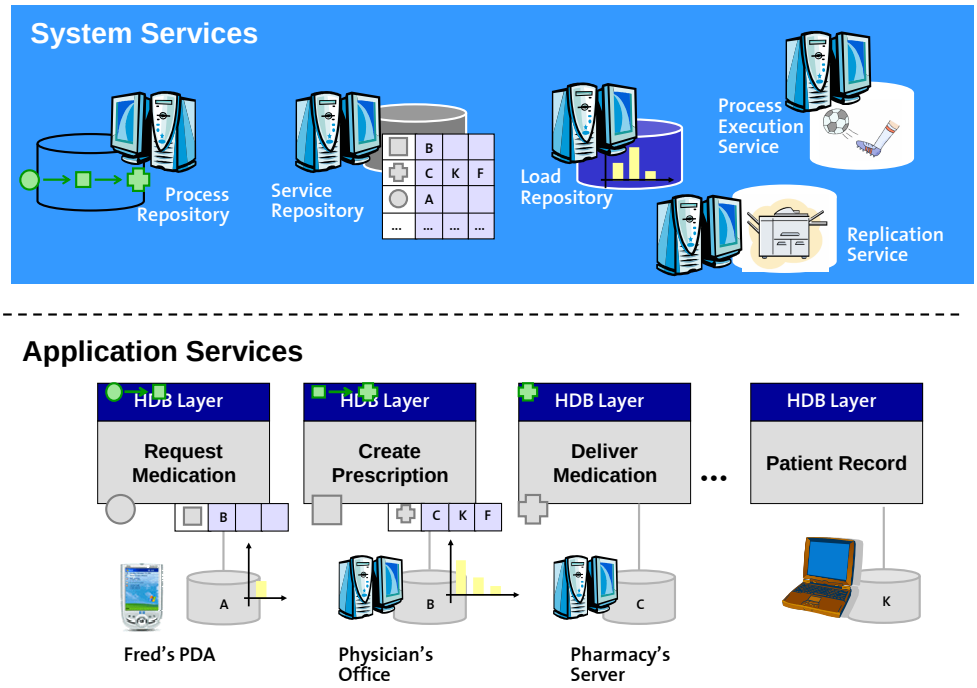


Figure 3.2: The Hyperdatabase Architecture

Fig. 3.3 illustrates how process metadata is generated, stored, and replicated in the distributed HDB network. After the design of the example process specification (see Fig. 3.1), the process is uploaded to the *process repository*. For illustration purpose, we use the same green symbols to illustrate the different service invocations within the process. After upload, the process definition stored in the process repository is chopped into smaller pieces, so called *execution units*, for replication. Each execution unit contains enough information for a participating peer to fulfil the task locally and to know the immediate next activities within the control flow of the process. Conceptual, the execution unit corresponds to an edge in the control flow of the process. Given the example process of Fig. 3.1, the HDB-layer of Fred's PDA has to know that after execution of Fred's request the process instances has to be passed to Fred's physician for creation of a prescription. The HDB-layer of the physicians computer has to know that after performing the prescription the pharmacy has to be contacted for delivery. Finally, the HDB-layer of phar-

macy's server has to know that the deliver activity is the last one and after successful execution of the delivery service the process execution has finished gracefully.

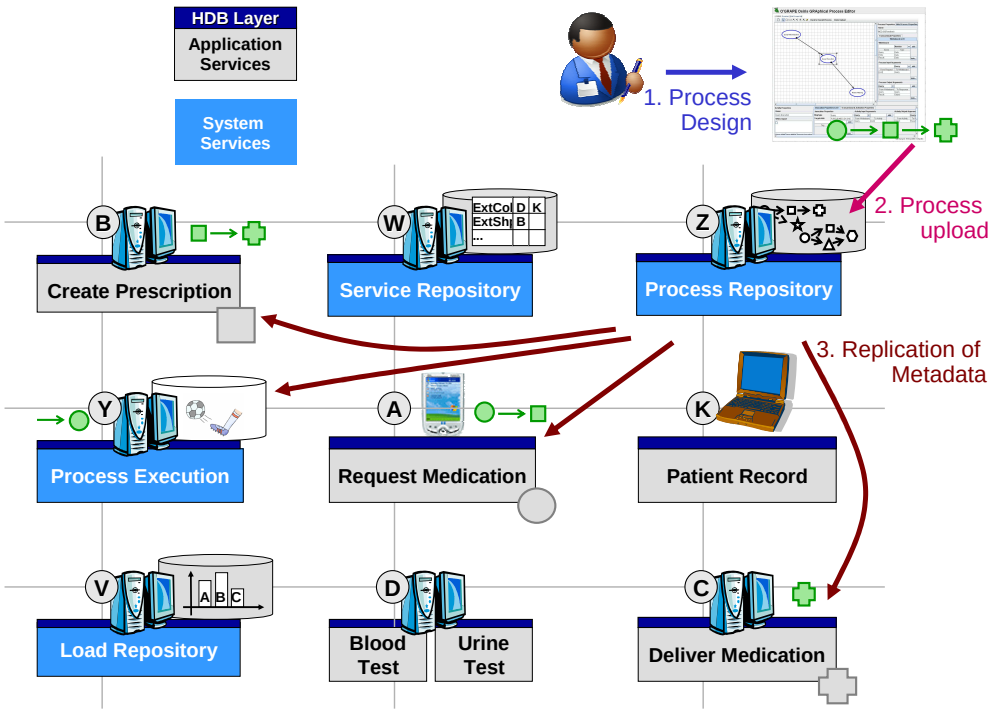


Figure 3.3: HDB Metadata Replication

The metadata replication reduces dramatically the probability that this information has to be requested at runtime. From another perspective, the local HDB-layers perform their tasks based mainly on local versions of the global meta information. Process execution is always triggered by an instance of the process execution system service. Process execution services get metadata about the first activities to execute within a process. For this reason the process execution services creates and migrates a process instance to a peer providing one of these services. Furthermore, during process execution a peer works off its part of a process and then directly migrates the process instance data to nodes offering a suitable service for one of the next steps in the process. For this reason, process navigation has to be decoupled from metadata replication and allows for efficient distributed process execution. Moreover, it

is often sufficient to hold approximate versions of metadata: for instance, load information of peers is required by the local HDB-layer to balance the process workload among the available service providers. Since this is an optimization, it is sufficient if local load information is only approximately accurate. Note that this would not lead to incorrect execution. In worst case, the load distribution among all service providers is just not optimal. On the other hand, changes on process definitions have to be propagated immediately to be available for new process requests. However, even this information can be replicated in a lazy way, since process instances are explicitly bound to a dedicated version of a process definition.

3.1.2 Basic Functionality of the Hyperdatabase System

In the following, we describe the most important functionality of a Hyperdatabase system with respect to decentralized peer-to-peer process execution.

- **Messaging:** The HDB-layer allows for a reliable exchange of messages between all nodes of the HDB network.

- **Service Provider:** The HDB-layer is able to host both system and application services and offers life-cycle management of service instances. Life-cycle management includes activation, control, and deactivation of services. Control of services includes important tasks, like checking the current load of services and verification of correct service execution.

- **Global Repositories:** Host global meta information of the infrastructures. The most prominent examples are process definitions, available services and their providers, and current load of providers.

- **Meta-data Replication:** The HDB-layer is responsible for meta-data replication as needed for peer-to-peer process execution. Meta-data replication is based on a hierarchical organization with global repositories and clients replicating from them. The replication is keeping local replicas consistent.

- **Process Design:** The hyperdatabase offers a graphical process design tool which allows non-programmers to design new application processes in a boxes and arrows approach.

- **Process Execution:** The HDB-layer is in charge of local process execution. Since replication of meta-data guarantees that a runtime enough

information for process execution is locally available, the HDB-layer receives process instances, invokes local service calls as part of the process. After that, the running process instances are routed to the next best-suitable HDB-nodes.

3.2 Extending the Hyperdatabase for Data Streams

In this section, we describe the proposed extensions to the Hyperdatabase concept in order to achieve flexibility and reliability for DSM with particular respect to monitoring applications in healthcare. Stream processing within a stream-enabled Hyperdatabase system is done by execution of *stream processes* in order to process *data streams*.

3.2.1 Data Streams

A data stream is defined as a continuous transmission of data elements. Each data element contains several data items as payload information, e.g., sensor readings like Fred's current blood pressure. These data elements have a discrete time, which is realized by designated data items containing sequence numbers relative to the order of the data element within the stream. This discrete timestamp allows for recognition of missing data elements and correct ordering of elements within a data stream. Additionally, data elements have a continuous global timestamp. The continuous global timestamp allows for timely correlation between data elements of different data streams. Particularly for the intended healthcare application scenario, this is very relevant since we want to analyze different physiological signs of Fred, e.g., heart activity and blood pressure, in combination. Due to temporal correlation of these different signs, medical relevant information can be extracted. Moreover, there is no restriction on the amount of data elements within a data stream. For this reason, data streams are theoretically unlimited. Therefore, it is in general impossible to store a "complete" data stream and process it in a single discrete event. Due to these constraints, data stream processing results have to be produced continuously and the results always correspond to a limited time-frame of input data streams. In addition, applications restrict the allowed processing delay between receiving input data stream elements coming from sensor devices and producing corresponding result data stream elements containing relevant extracted information. For example, for the detection of critical health conditions the delay between the critical condition happening in Fred's body, e.g., a heart attack, and the triggering of an emergency service is of high relevance.

3.2.2 Stream Processes vs. Workflow Processes

Stream processes are a composition of *data stream operators*, or shortly operators, interconnected by data streams. Services or activities in workflow processes of workflow management are discrete invocations in request/response style. Usually, these invocation are rather short events. Contrarily, the operator of a stream process perform continuously running operations on the incoming data stream elements by producing outgoing data stream elements. Moreover, the operators are continuously processing with respect to current and the previously seen data stream elements and therefore keeping an continuously changing *operator state*. Contrarily, services within workflow processes are usually stateless. Although, recent work in the area of Web-services and Grid computing developed also standards for stateful Web-services, like the *Web Service Resource Framework (WSRF)* [HWMB04]. Based on this, work has been done to use the WSRF standard directly for DSM [LL04]. But we have not followed this heavyweight approach because we state that the WSRF standard is not suitable for DSM and by far not sufficient. In particular considering the requirements on DSM imposed by the intended application areas, like e.g., support of mobile devices.

Comparing the two kinds of processes with respect to control and data flow, we see that for workflow processes the control flow is dominating and therefore explicitly modeled by the process designer as edges. In this case, the order of service invocations is vital for the correct execution of the process. Also the data flow is modeled by the process designer with the use of whiteboard and mappings to and from activities. Contrarily, stream processes are dominated by data flow. The control flow of a stream process is not obvious because all operators are continuously running. Nevertheless since no application is running forever, we can define a life-time of a stream process consisting of three phases. First, during the *activation phase* all necessary operator instances and their interconnecting data streams are established by the infrastructure. Second, during the *runtime phase* all operators are up, continuously running, and passing the data flow via data streams. Third, during the *deactivation phase* all running operator instances and so the stream process instance are gracefully stopped. We can only define a control flow for the activation and deactivation of a stream process, where discrete activities, i.e., the activation or shutdown of an operator instance, are performed. For this reason, workflow processes are used to activate and deactivate a stream process instance.

Figure 3.4 illustrates an example process taken from our telemonitoring scenario (see Section 2.2). We have already presented this stream process on the right hand side of Fig. 2.1 in Section 2.1.1. This stream process is supposed to monitor heart activity and blood pressure of patient Fred. The “ECG

Acquisition" and "Blood Pressure Acquisition" operators are executed at a mobile device, e.g., a PDA or smartphone, carried by the patient. In order to gain more relevant medical information about Fred's health condition the variability of the heart rate and the blood pressure is jointly analyzed. This operator may be executed at the base station in Fred's home. Both, the "Critical Detection" operator and the "Analysis" operator are joining the two data streams of blood pressure and heart activity. The "Critical Detection" is executed at Fred's base station at home and is checking whether some critical condition arise which need immediate intervention. The "Analysis" is executed at Fred's caregiver and does some long-term and more thorough joint analysis in order to give Fred's physician valuable information for therapy control.

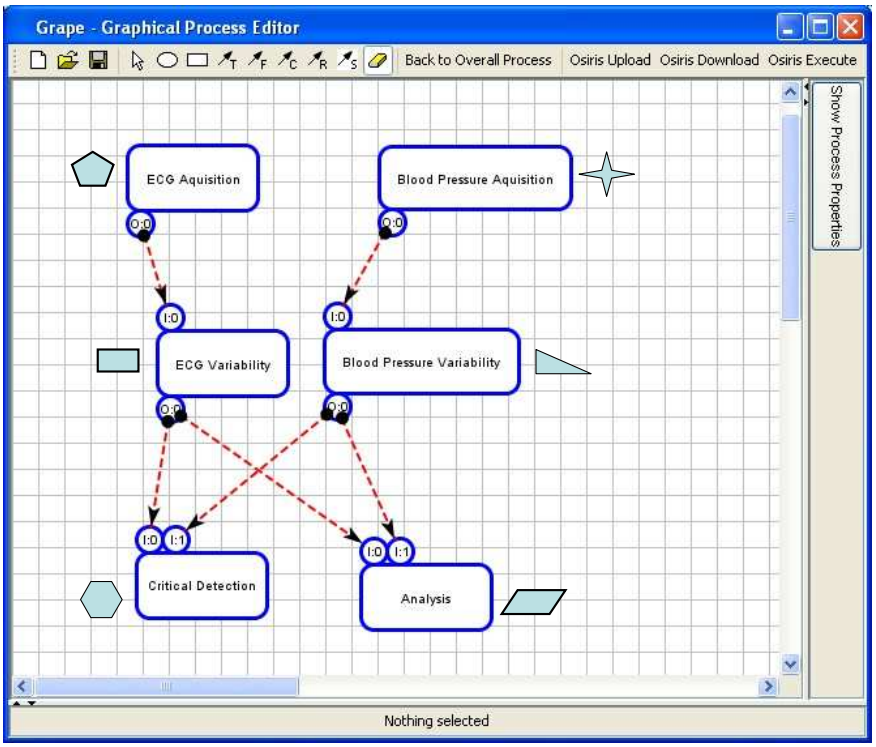


Figure 3.4: An Example Stream Process Definition

As Figure 3.4 also illustrates, the process design tool O'GRAPE [WSN+03] has been extended in order to model stream processes. For the modeling of data stream operators, the notion of *ports* has been introduced. Ports may either be for input or output and allow the consumption or production of a data

stream, respectively. In order, to design a valid stream process all output ports of operators have to be connected with at least one input port ports by a data stream. Since there is a strict consumer/producer pattern along a data stream multiple different consumers are allowed, hence one output port is allowed to feed multiple input ports. If ports of operators are not used for a certain stream process they can be explicitly disabled. Within O'GRAPE input ports are drawn as small circles attached above the operator symbol and output ports are small circles attached below the operator symbol. Obviously, each input and output port needs a corresponding numeric ID for identification. The data streams connecting output and input ports are drawn as red dashed line between the according ports.

For meta-data replication, stream process definitions are decomposed into pairs of subsequent operators, also *transmission units (TU)* similar as for workflow process definitions where pairs of subsequent activities are called *execution units*. Contrarily to execution units which model the control flow, the TU's model the stream data flow.

We consider stream processes in a similar way as workflow processes with respect to important aspects like distributed execution, load balancing, or fault tolerance. As for workflow management, we need for DSM to distribute meta information about stream processes. For this task, replication management of the Hyperdatabase can be exploited. Replication management is extended by enabling global repositories to hold necessary information to establish and run stream processes (like information about stream process definitions, providers of operators, operator load, backup information, etc.). Also the HDB-layer has to provide additional functionality, like transport of data streams, activation, deactivation and routing of stream processes, execution of operators, load and failure handling for stream processes, to mention the most important. Subsequently, we describe the basic architecture for a DSM enabled Hyperdatabase system.

3.2.3 Additional HDB-Functionality for DSM

- **Stream Transport:** The extended HDB-layer allows for a reliable transmission of data streams. Reliable transmission of data streams include the detection of gaps or duplicates in data stream elements and to guarantee *first-in first-out* (FIFO) transport of data stream elements to keep the correct order of the elements within a data stream.

- **Stream Operator Management:** The extended HDB-layer is able to host both instances of stream operators and offers also life-cycle management of running operator instances. Life-cycle management includes

activation, control, and deactivation of operator instances. The control includes important tasks, like checking the current load of services and verification of operator execution. Verification includes reliability algorithms that are described in Chapter 5.

- **Extended Global Repositories:** The global repositories have been extended in order to support meta-data about stream process definitions. Moreover, also information about available types of operators and the corresponding providers needs to be replicated. In order to support, checkpointing of running operator instances as needed for the presented reliability algorithms in Chapter 5 also the information about the current backup provider is new meta-information which needs to be stored in a global repository.

- **Meta-data Replication:** The functionality of the basic HDB-system is used without conceptual changes.

- **Stream Process Design:** The graphical design tool of the hyperdatabase has to be extended in order to support design of stream process. In particular, the design of operators, their corresponding ports and connecting data streams has been added to the process design tool.

- **Reliable Stream Process Execution:** The HDB-layer has been extended to support the reliable execution of stream processes. For this reason, the extended HDB-layer is able to start, stop and control running operator instances and connect their data streams appropriately. Moreover in order to support reliable execution, running operator instances are subject of *checkpointing*. Checkpointing mechanism keeps a backup of current operator state at a backup provider consistent so that in case of a failure the operator instance can be recovered and executed at the backup provider (for details see Chapter 5). This mechanism is called *operator migration*.

As defined by the hyperdatabase concept, the global repositories do *not* require a *centralized implementation*. If the number of nodes in the hyperdatabase network is increasing, additional repositories of the same kind can reduce the load of repositories (i.e., by applying horizontal partitioning). The different physical repositories form again a single global repository. Efficient horizontal distribution of load can be achieved by applying Peer-to-Peer concepts, like *distributed hash-tables (DHT)* [SMLN+03, ZKJ02]. But this topic is out of scope of this thesis.

Figure 3.5 illustrates the architecture of the extended HDB infrastructure for DSM processing. The example stream process of Fig. 3.4 is uploaded to a

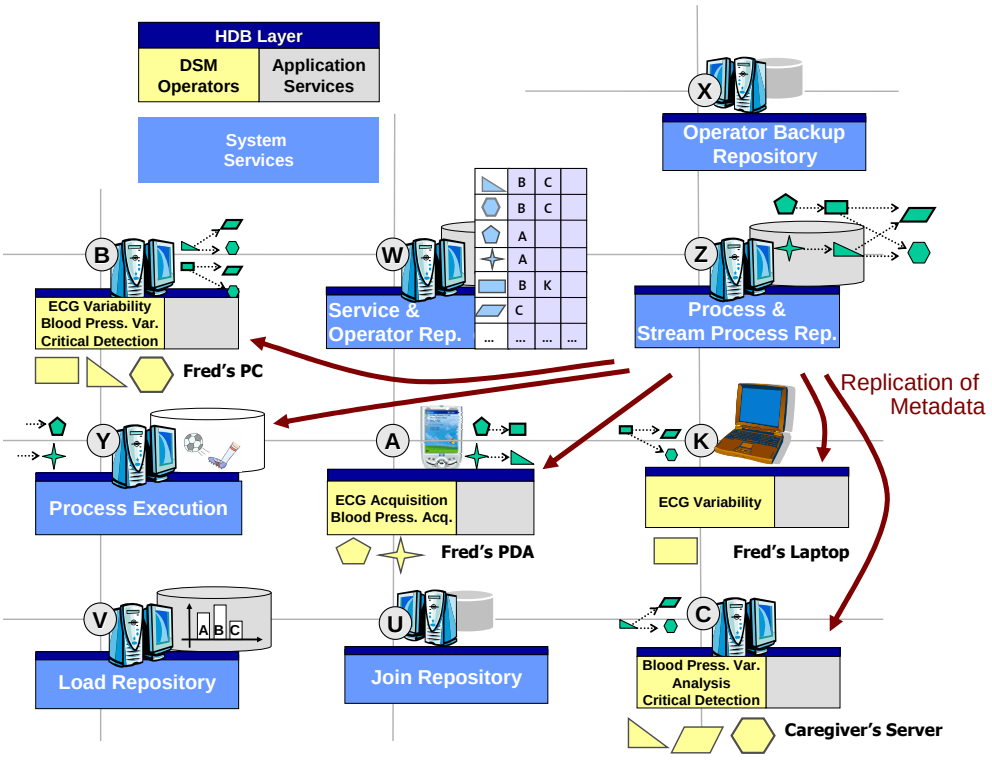


Figure 3.5: The Extended Hyperdatabase Architecture

global repository for stream process definitions. Based on chopping the stream process definition in transmission units, the TU's are replicated at the relevant peers. This mechanism allows similar to workflow processes, the true peer-to-peer execution of stream processes without contacting global services during runtime. For illustration, we describe how Fred's monitoring applications is using the extended HDB. Fred's physician has designed the stream process (according to Fig. 3.4) and has also adjusted Fred's operators to his individual parameters (e.g., thresholds for the critical detection). After applying the necessary body sensors to Fred, the stream process is invoked, by sending activation messages for the sensor operators to Fred's PDA. Beginning from there, all subsequent operators (on Fred's PC and caregiver's PC) are activated in peer-to-peer style. For now, all operators are up and continuously processing data streams. In case of critical conditions (e.g., Fred has a heart attack), his physician has designed appropriate processes (e.g., calling the emergency service) which are invoked by the local HDB-layer if

necessary. Snapshots of internal operator states are saved regularly on an operator backup device for recovery (e.g., Fred's PC for operators running on Fred's PDA and vice versa). If Fred's PDA fails (e.g. due to empty batteries), operators running on Fred's PDA are migrated to Fred's PC and are initialized with the recently saved internal states. Now, Fred's PC is receiving data directly from the wireless body sensors, because it hosts the acquisition operators. Fred is required to stay near his PC, due to limited transmission range of the body sensors. When Fred is out of range, the HDB-layer on Fred's PC is able to invoke processes to deal with this situation (e.g., inform Fred to come back in range or stop the streaming process gracefully). Because of our peer-to-peer approach processes may be executed off-line without central control. When Fred is replacing the empty batteries of his PDA, the operators are migrated back and full functionality is available again without loss of data.

As shown in this chapter, our proposed information management infrastructure will incorporate DSM into process management of HDB-systems in peer-to-peer style. Our solution offers great flexibility for healthcare applications. Non-programmer users are able to define individual stream processes for different monitoring applications. Additionally, processes for failure handling, result delivery, and critical event handling are supported. Our system is extendable to new applications (i.e., diseases) by adding new operator elements (e.g., an additional glucose sensor if Fred will suffer from diabetes in the future). Also reliability on system level is provided by the extended HDB. Whenever a node fails, this is handled by node recovery or, if not possible, by operator migration. Details on operator migration and reliability of DSM are discussed in Chapter 5. In case operator migration is not available, user defined failure processes (both streaming and workflow) may be invoked. More details on failure handling of the DSM infrastructure is discussed in Chapter 5. In addition, workflow processes can affect stream processes. For example, a process called "Fred leaves the house" can stop DSM gracefully. In general, this combination permits more complex applications, where process and data stream management work seamlessly together.

4

Data Stream Management Model

In the following, we describe a formal model of a data stream management system. The *DSM system* (DSMS) coordinates the execution *stream processes* on top of data stream operators in a network of loosely coupled *hosts* and ensures correctness even in case of failures.

4.1 Basic Data Stream Model

The presented formal model will use mathematical notation taken from *set theory* where capital letters are used to denote sets. Using a hat on top of a capital letter referencing to a set indicates that the set is global and therefore containing all available objects within our model. For this reason, $\widehat{\mathsf{H}}$ denotes the finite set of all hosts participating within the DSM system. Since usually $|\widehat{\mathsf{H}}| > 1$, a DSMS is a distributed system. Each host is able to perform certain data stream operations, also called *operator types*. The finite set of all operator types available for execution within the DSM system is called $\widehat{\mathsf{OT}}$. The subset of all operator types available at a given host $\mathsf{h} \in \widehat{\mathsf{H}}$ is called $\mathsf{OT}(\mathsf{h})$. Of course, hosts may join and leave the DSMS at any time since we are assuming a *dynamic* system. For the ease of notation, we have omitted the time dependency for participating hosts and their operators.

Stream process definitions combine operator types to build up complex stream processing tasks which consume and produce data streams of/for the *outside world* which encompasses all external systems interacting with the DSMS. The DSMS is executing instances of stream process definitions, called *stream processes*. Stream processes make use of instances of operator types, also called *operators*. Operators are continuously processing *data streams*. In the following, we introduce these terms with formal definitions.

4.1.1 Data Stream Management System (DSMS)

Definition 4.1 A *DSM system* (DSMS) is defined as the following 4-tuple:

$$\mathcal{DSMS} = \left\langle \widehat{\mathsf{H}}, \widehat{\mathsf{OT}}, \widehat{\mathsf{SPD}}, \widehat{\mathsf{SP}} \right\rangle$$

where $\widehat{\mathsf{H}}$ is a set of hosts participating in the DSM system, $\widehat{\mathsf{OT}}$ the set of global available operator types offered by all hosts, $\widehat{\mathsf{SPD}}$ is a set of available stream process definitions, and $\widehat{\mathsf{SP}}$ is the set of running stream process instances. ◇

Of course, the participating hosts $\widehat{\mathsf{H}}$, the available operator types $\widehat{\mathsf{OT}}$, the available stream process definitions $\widehat{\mathsf{SPD}}$, and the running stream process instances $\widehat{\mathsf{SP}}$ are not static but vary over time within the DSMS.

4.1.2 Data Streams and Data Stream Elements

Definition 4.2 A *data stream* DS is a possibly infinite, totally ordered set $(\mathsf{DE}, \prec, \mathsf{op}, \mathsf{ip}, \Sigma)$ of data stream elements $\mathsf{de} \in \mathsf{DE}$. DE is the set of all elements within a stream. The connection point of a data stream at producer-side is called *output-port* op and at consumer-side *input-port* ip. The input/output naming is assigned from a producer/consumer point of view. The symbols to be sent as payload within data stream elements are defined in the *data stream alphabet* Σ. ◇

A DS represents a continuous transmission of data stream elements de in a temporal order between a producer and a consumer. The notation $\mathsf{DS.ip}$ refers to the input-port of the data stream DS and $\mathsf{DS.op}$ refers to the output-port, respectively.

Definition 4.3 A data stream element $\mathsf{de} \in \mathsf{DS}$ is defined by the following tuple: $\mathsf{de} = \langle \tau, \xi, \mathsf{pd} \rangle$ where $\tau \in \mathrm{R}^+$ is a global timestamp attribute which is given to a data stream element at the time of processing, $\xi \in \mathrm{N}_0$ is a sequence number which is used for ordering and gap detection of the data stream elements within the data stream, and pd is the payload information, which is a symbol of the data stream alphabet $\mathsf{pd} \in \Sigma$. ◇

A data stream element shows some similarities to a tuple of a relational database table like being structured according to a given schema. Therefore, both are *structured objects*. But they differ in their cardinality. Whereas a relational table has a finite number of elements, the number of data stream elements in a data stream is potentially unlimited. In order to ease the notation of structured objects and their attributes in this thesis, we are using a labeled notation to refer to attributes as presented in the original work of

Codd [Cod70] about relational algebra. This has the advantage of being independent of an ordering of attributes and having a more compact and concise representation. For example, the payload attribute pd of a data stream element de is denoted in this thesis as $de.pd$.

Note that, the data stream type is corresponding to the data stream alphabet. Different types of data stream elements use different alphabets for encoding the payload information. Within a data stream all containing data stream elements are of the same type:

$$\forall de \in DS_a : de.pd \in DS_a.\Sigma$$

According to the previous definition data stream elements have two temporal ordering contexts. Firstly a global timestamp, called *processing time*, and secondly a logical sequence number, called *stream time*. Whenever we investigate a temporal portion of a data stream we can apply the following subsets for processing time and stream time:

Subset of DS before time κ_1:

$$DS(\kappa_1) \subseteq DS|\kappa \leq \kappa_1$$

Subset of DS within a time interval κ_1, κ_2:

$$DS(\kappa_1, \kappa_2) \subseteq DS|\kappa_1 \leq \kappa \leq \kappa_2$$

Where $\kappa = \tau \in R^+$ for processing time or $\kappa = \xi \in N_0$ for stream time.

For example, a data stream element coming from a sensor and arriving at a processing operator may look like the following:

$de_{189} = \langle$
29.02.2008 10:00:00.50,
189,
$<sample>$
$<time>29.02.2008\ 10:00:00.00</time>$
$<value>10mV</value>$
$</sample>$
$\rangle$

The data stream element is the 189^{th} element within the data stream (stream time). It was received by the operator at 10:00:00.50 of the 29^{th} of February 2008 global time (processing time). The payload of the element contains an structured entry, e.g., in XML format. The payload was generated by the sensor and contains the global time of generation and the value 10mV (e.g., a sensor voltage value that corresponds to a physical quantity). Moreover, this example illustrates that processing time and stream time give no

information about the original time the data stream element was generated by the sensor (*generation time*). If this information is necessary for the various applications (e.g., telemonitoring in healthcare) it must be part of the payload information. Of course, if data streams coming from different sensors are jointly processed on the basis of generation time the clocks of the different sensors have to be synchronized. Such synchronization issues are out of scope of this thesis.

4.1.3 Operator Type (OT)

Based on the data stream definition, we define *operator types* which are in charge of processing data stream elements of incoming data streams and producing derived data stream elements of outgoing data streams. The connection points of data streams to the operator type are called *ports*. Each operator type ot has an ordered set of $n \in N_0$ input-ports $IP(ot)$ with cardinality $|IP(ot)| = n$ and an ordered set of output-ports $OP(ot)$ with cardinality $|OP(ot)| = m$. Each output-port $op_i \in OP$ produces data stream elements with the stream alphabet $op_i.\Sigma$. Accordingly, each input-port $ip_i \in IP$ expects to receive data stream elements with the stream alphabet $ip_i.\Sigma$. The behavior of the operator type is deterministic and can be modeled as finite state machine (FSM). Figure 4.1 is illustrating the operator type model. Two streams shown in Fig. 4.1 have a special purpose. Firstly, the *config stream* allows for changing of the operator state explicitly. This is in particular useful for operator initialization. As an example, we present an operator processing medical alerts of our patient Fred. The operator counts the number of medical alerts of Fred since the beginning of the measurement and the number of medical alerts in the last hour. For this reason, the operator may have the following internal state (as XML representation):

```
< State >
 < BeginOfMeasurement >
  29.02.2008 10:00:00
 < /BeginOfMeasurement >
 < AlertsOverall >
  16
 < /AlertsOverall >
 < AlertsLastHour >
  1
 < /AlertsLastHour >
 < StartOfCurrentHour >
  29.02.2008 18:00:00
 < /StartOfCurrentHour >
```

$< /State >$

An operator instance may not be initialized with an empty state but with an already existing state. For example, due to a failure in processing the operator instance has to be restarted at a different host node. The config stream data element that initializes the restarted operator instance on the 29^{th} of February 2008 at 18:10:00.00 is:

$de_1 = \langle$
29.02.2008 18:10:00.00,
1,
$< State >$
$< BeginOfMeasurement >$
29.02.2008 10:00:00
$< /BeginOfMeasurement >$
$< AlertsOverall >$
16
$< /AlertsOverall >$
$< AlertsLastHour >$
1
$< /AlertsLastHour >$
$< StartOfCurrentHour >$
29.02.2008 18:00:00
$< /StartOfCurrentHour >$
$< /State >$
$\rangle$

Secondly, the *state backup stream* allows for reading the current operator state. This allows to take checkpoints of operators during execution in order to achieve reliability. More on reliability of operator execution will follow in Chapter 5. Elements of both of this data streams are structured as any other data stream elements containing processing time, stream time, and payload information.

Definition 4.4 An *operator type* OT is defined by the following tuple:

$$OT = \langle \Theta, \Gamma, ST, \delta, \omega, IP, OP, min \triangle \tau \rangle$$

Where Θ is the input alphabet as cartesian product over the alphabets of all n data streams received by the ordered set of input-ports IP:

$$\Theta = \prod_{i=0..(n-1)} (\Sigma_i^{in} \cup \{\lambda\})$$

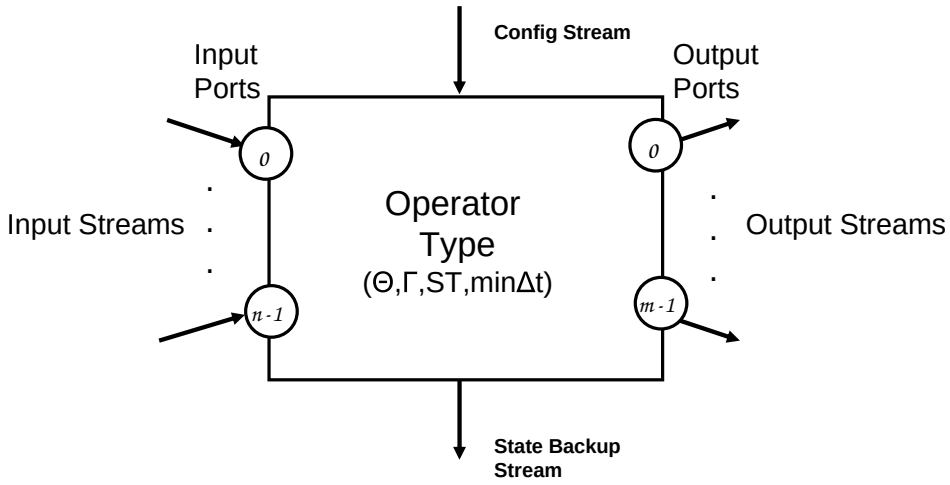


Figure 4.1: Operator Type Model

Γ is the output alphabet as cartesian product over the alphabets of all m data streams produced by the ordered set of output-ports OP:

$$\Gamma = \prod_{i=0..(m-1)} (\Sigma_i^{out} \cup \{\lambda\})$$

ST is a finite, non empty set of operator states. The state transition function δ is defined as:

$$\delta : ST \times \Theta \rightarrow ST$$

The output function ω is defined as:

$$\omega : ST \times \Theta \rightarrow \Gamma$$

$min \triangle \tau \in R^+$ is describing a minimal delay for processing a single state change of this operator type. $\diamond$

According to this definition, an operator type describes a deterministic finite state machine. Whenever new input data stream elements are available, δ is applied in order to proceed to the new state and ω is applied in order to produce output data stream elements.

The λ symbol identifies input or output data streams where no data stream element is present. In this model, the FSM works asynchronously which means that the FSM has not to wait for data stream elements to be present at each input in order to proceed to the next state. Synchronous operator types can be emulated inside the operator type implementation (e.g. by modeling a state transition to the same state if the the input is not complete). Similarly, not every state change has to produce data stream elements at each output.

4.1.4 Stream Process Definition (SPD)

Based on the two basic DSM building blocks of operator types and data streams, we are now able to combine these two in order to define complex stream processing tasks, called *stream process definitions*. A stream process definition is described as a graph, where the vertices are operator types and the edges are data streams. Since a data stream has a direction of data flow, the graph is also directed. The graph is a multigraph because there can be more than one edge (data stream) connecting a pair of vertices (operator types). For example, we consider a combined heart and blood pressure sensor, which is generating a heart activity and a blood pressure data stream as two distinct output data streams and an analysis operator type which is combining blood pressure and heart activity from two distinct input data streams. In this case, we have two edges (data streams) from the combined sensor to the analysis operator type.

Definition 4.5 A *stream process definition*, SPD is defined by the following tuple:

$$\mathsf{SPD} = \langle V, E \rangle$$

SPD is describing a directed multigraph, where V is a finite set of vertices and E is a finite set of edges. $\diamond$

Definition 4.6 The set of vertices V of SPD is defined as:

$$V = \{\langle ot, s_o, \triangle\tau\rangle \,|ot \in \{\widehat{OT} \cup \{ow\}\}, s_0 \in ST, \triangle\tau \in R^+\}$$

where ot is the describing the operator type of the vertex. Additionally to the operator types offered by the DSMS, outside world interactions are represented by ow-vertices. The initial state of the vertex using the operator type

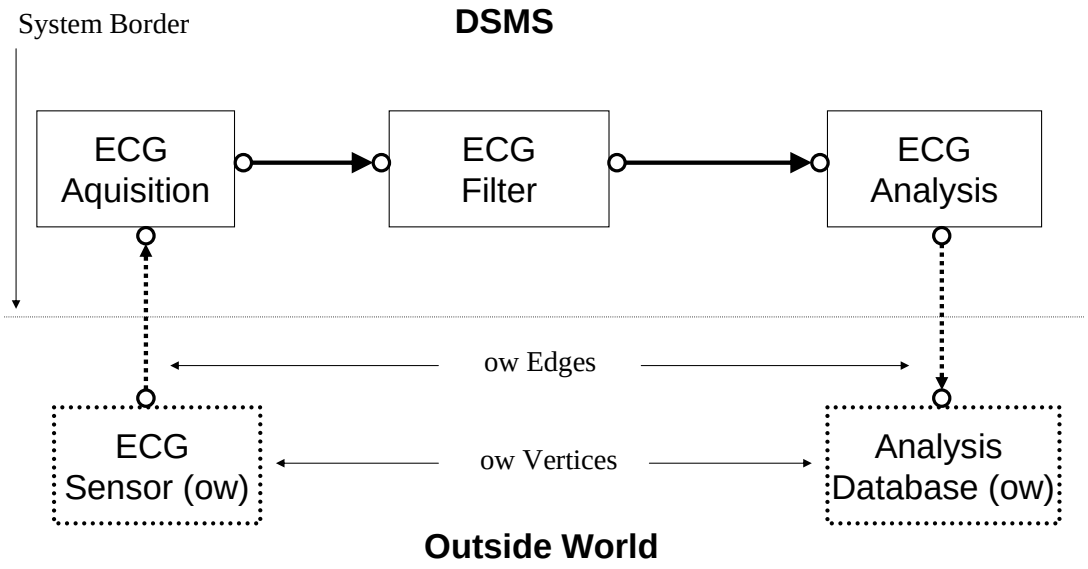


Figure 4.2: Example Stream Process Definition

is given by s_0, this state is set by sending along the *config stream* of the operator instance at the time of startup. A detailed description on this will follow in Section 4.1.5. In order to specify a maximum tolerable delay for processing a state transition at this vertex, $\triangle\tau$ is given. ◇

Definition 4.7 The set of edges E of SPD is defined as:

$$E = \{\langle x, op, y, ip, \triangle\tau\rangle \,|x \in V, op \in x.ot.OP, y \in V, ip \in y.ot.IP, \triangle\tau \in R^+\}$$

where x is the source vertex, op is the corresponding output port at the operator type ot of vertex x. Similarly, y is the destination vertex, ip is the corresponding input port at the operator type ot of vertex y. Edges where either the source vertex or the destination vertex is an ow-vertex are called ow-edges. In order to specify a maximum tolerable transfer delay for a data stream element along this edge, $\triangle\tau$ is given. ◇

This definition does not give any constraint on the correctness of a stream process definition. For example, a *well-formed* stream process definition has to have for each input port of a vertex exactly one input edge attached to it. For details on this see Section 4.3.

Fig. 4.2 illustrates a simple example stream process definition taken from our example telemonitoring scenario (see Section 2.2) by applying the presented DSM model. The stream process offers heart activity analysis by applying three operators. The first operator is reading out the sensor device. The second operator is applying filtering a signal level in order to remove noise. Finally, the third operator is deriving medical relevant information of the *electrocardiogram (ECG)*. Consequently, the example consists of three vertices where $\mathrm{ot} \in \widehat{\mathrm{OT}}$ and two outside world vertices where $\mathrm{ot} = \mathrm{o}w$. The sensor device is an $\mathrm{o}w$-vertex. The acquisition is already part of the DSMS. Furthermore filtering and analysis of the heart signal is performed. Finally, the DSMS is producing an data stream feeding a analysis database, which is an $\mathrm{o}w$-vertex again.

4.1.5 Stream Process Execution

When during the execution of a stream process a running instance of an operator type is generated at a host, this instance is called *operator*. Additionally to the operator type, the operator has a current time context, a current state and a current host. Fig. 4.3 illustrates the operator model.

Definition 4.8 An *operator* O is a running instance of a vertex of a stream process and has the following definition:

$$\mathrm{O} = \langle v, \Xi^{\mathrm{in}}, \Xi^{\mathrm{out}}, s, \mathrm{h}, \tau \rangle$$

where $v \in \mathrm{SPD.V}$ is a vertex taken from the corresponding SPD, Ξ^{in} is the set of stream times of last $\mathrm{d}e$ processed at each input port. Analogous, Ξ^{out} is the set of stream times of the last $\mathrm{d}e$ processed at each output port. The current state of the operator is given by $s \in \mathrm{ST}$, the host executing the operator instance is $\mathrm{h} \in \widehat{\mathrm{H}}$, and the current global time is indicated by τ. $\diamond$

In order to initialize running operator instances a data stream element with a new initial state as payload can be sent to the operator instance along the config stream. In order to retrieve the current state of an operator during execution, the state backup stream produces a data stream element with the current state as payload information for each state transition of the operator. Whether this data stream or others are retrieved by push or pull technique is not described in this model and therefore part of an implementation.

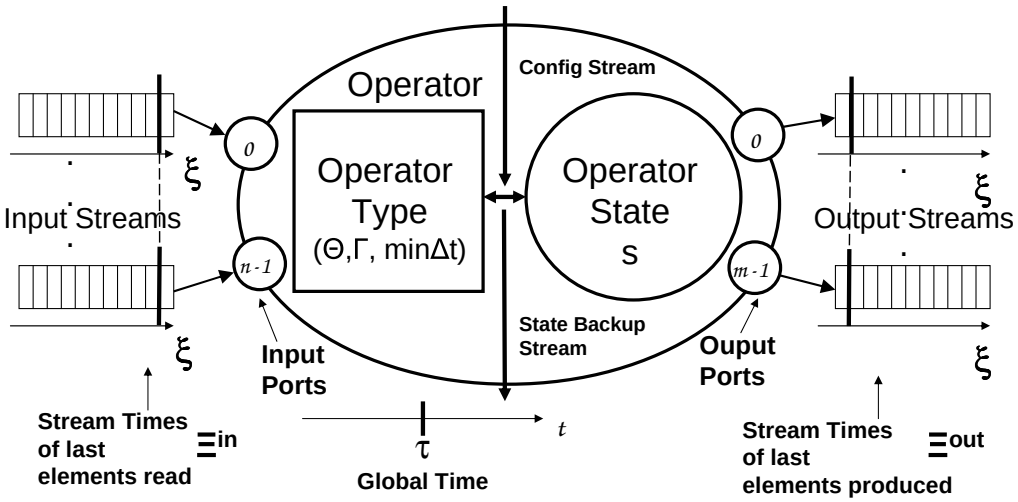


Figure 4.3: Operator Model

Finally, running operator instances form together a running instance of a stream process definition, called *stream process*. Additionally, to the stream process definition the stream process has a set of operator instances. In this set of operators, we do not consider outside world o_w-vertices because this operator instances are not executed within the modeled DSM system (DSMS).

Definition 4.9 A *stream process*, SP is defined by the following tuple:

$$\mathrm{SP} = \langle \mathrm{SPD}, \mathrm{OS}, \mathrm{DSS} \rangle$$

where SPD is the corresponding stream process definition and OS is a set of operators executing the operator types given by SPD. DSS is a set of data streams connecting the operator instances ◇

This definition does not give any constraint on the correct activation of a stream process instance. For example, a *well-activated* stream process has to

have a running operator instance in OS for each non-*ow* vertex in the stream process definition. For details on this see Section 4.4.

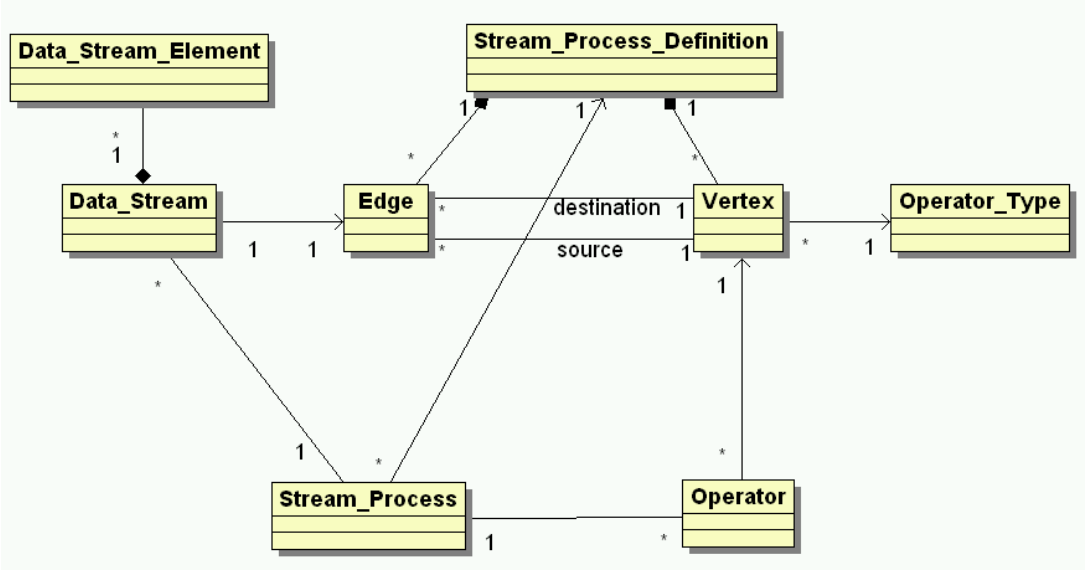


Figure 4.4: UML Diagramm of the DSM Model

Figure 4.4 overviews and shows relations between the different entities of the presented DSM model by use of an UML class diagramm.

4.2 Outside World Interactions

As stated in the stream process definition, the DSMS is interacting with the outside world (*ow*-vertices). From the outside world point of view, the DSMS is seen as black box, which is interacting with the outside world through data streams as illustrated in Fig. 4.5. These data streams are called *outside data streams* or *ow*-edges.

In this thesis, we assume for our DSM model the outside world system *does not fail*. Of course, in real world the outside world system may fail, but from the DSM system point of view there is by nature no possibility to cope with all kind of failures that result from this without performing active

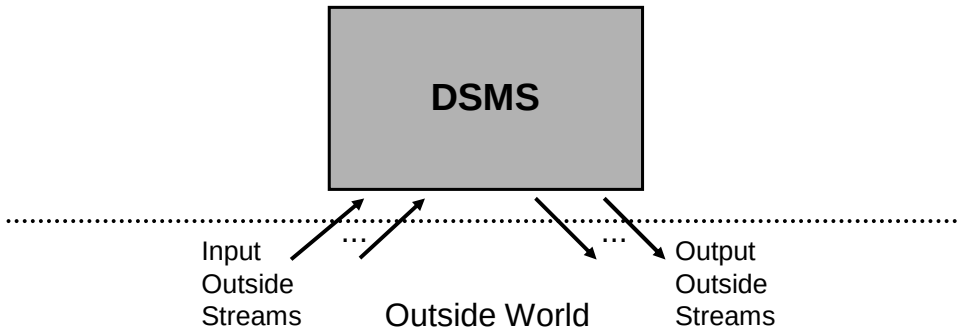


Figure 4.5: Outside World

control on the outside world. For illustration, we take an example from our telemonitoring scenario. Let us assume Fred's heart activity is monitored by a DSM system. For this reason, Fred has sensor hardware attached to his body which is measuring the heart activity with electrodes attached to his breast. The sensor hardware wirelessly transmits the sensor data stream to Fred's PDA. In this scenario, we consider the PDA as node of the DSM system, but the sensor hardware is part of outside world system. If the sensor hardware fails, the DSMS is no longer able to continue correct DSM monitoring of Fred's heart because the incoming *ow*-edge is dead. Surely, there is the possibility to cope with this problem by having a redundant pair of sensor hardware. From a model point of view, applying this failure handling mechanism means that we move the system border between DSMS and the outside world in a way that the sensor hardware now is under control of the DSMS. On the other hand, e.g., if Fred's PDA fails, which is inside the DSMS, the operator instances executed there can be moved to Fred's smartphone, which is also inside the DSMS. In this case, the system border has not been moved.

Furthermore, we state two assumptions on the behavior of the outside world (o*w*-vertices) in order to make the failure handling of the DSMS more efficient. Here are the details of the two assumptions for input and output outside data streams:

Firstly, the outside world input is defined as receiving data stream elements from o*w*-vertices. For this case, we assume that the DSMS can *reread* data stream elements from the outside data stream if necessary, because of failures within the DSMS. Practically, this can be implemented e.g., by applying a reasonable buffer in the sensor hardware. This assumption is called *outside sender assumption* (OSA). Of course this assumption, only holds for a limited time-frame. But knowing this time limit and using this assumption allows the DSMS to apply more efficient reliability algorithms.

Secondly, the output is defined as sending data stream elements to o*w*-vertices. For this case, we assume that the DSMS is allowed to *resend* data stream elements to the outside data stream if necessary. This re-sending could be because of failures inside the DSMS. The practical implementation at the o*w*-vertex will eliminate duplicates of data stream elements. Duplicate elimination is allowed because our DSMS works strictly deterministic and therefore the duplicates are identical. Duplicates could be detected at the o*w*-vertex, e.g., by ignoring multiple data stream elements with the same stream-time (aka, sequence number). This assumption is called *outside receiver assumption* (ORA). Again, by leveraging this feature of the o*w*-vertex the DSMS can apply more efficient reliability algorithms.

4.3 Well Formed Stream Process Definitions

In this section, we describe the following set of constraints a stream process definition has to fulfill in order to be considered as valid or *well-formed*. In a practical system, this constraints can already be evaluated by the process design tools before process execution.

1. Each vertex in a stream process definition has attached to each input port *exactly one* input edge.

2. Each vertex in a stream process definition has attached to each output port *at least one* output edge.

3. The stream alphabet of each output port corresponds along all connected edges to the expected stream alphabet of input ports.

4. Outside world ow-edges are always connecting ow-vertices with non ow-vertices. The stream process definition is not describing interactions between ow-vertices.

5. Reasonable delay constraints are given within the stream process definition, which means the $min \triangle \tau$ constraints given by the operator type are not violated in the SPD.

Definition 4.10 A well-formed stream process (SPD) definition holds the following constraints:

$$\forall v \in SPD.V(\forall ip \in v.ot.IP(\exists! e \in SPD.E : e.ip = ip)) \quad (1)$$

$$\forall v \in SPD.V(\forall op \in v.ot.OP(\exists e \in SPD.E : e.op = op)) \quad (2)$$

$$\forall e \in SPD.E : e.op.\Sigma = e.ip.\Sigma \quad (3)$$

$$\nexists e \in SPD.E : (e.x = ow \wedge e.y = ow) \quad (4)$$

$$\forall v \in SPD.V : v.\ \triangle \tau \geq v.ot.min \triangle \tau \quad (5)$$

◇

Note that $\exists!$ describes the existence of exactly one element.

Finally, well formed stream process definitions may contain *cycles* in the graph. Cycles are *paths* within the stream process definition, where the start vertex corresponds to the end vertex. As defined in graph theory, a path is a sequence of vertices in a form that from each vertex there is an edge to the next vertex in the sequence. Nevertheless, we distinguish between cyclic and non-cyclic stream process definitions for certain aspects in the remainder of the thesis.

4.4 Well Activated Stream Process

After the design of a well-formed stream process definition SPD, the DSMS is in charge of activating an stream process instance SP of the given SPD. After successful activation, the following constraints on *well-activated* stream process instances are defined:

1. Each non ow-vertex in SPD has a corresponding running operator instance in OS.

2. Each edge in SPD has a corresponding running data stream instance in DSS.

Definition 4.11 A well-activated stream process (SP) holds the following constraints:

$$\forall v \in \mathrm{SP.SPD.V}|_{v \neq ow}(\exists! o \in \mathrm{SP.OS} : v = o.v) \quad (1)$$

$$\forall e \in \mathrm{SP.SPD.E}(\exists! ds \in \mathrm{SP.DSS} : (e.op = ds.op \wedge e.ip = ds.ip)) \quad (2)$$

$\diamond$

5

Reliable Data Stream Management

In this chapter, we describe how to apply reliable stream process execution within our proposed DSM infrastructure of Chapter 3. Due to the nature of DSM applications (e.g., as described in Chapter 2), the DSM system has to work in a distributed setting including mobile devices and wireless connection. Regarding this fact, efficiency of reliability algorithms have to be particularly considered.

In what follows, we firstly describe a model for reliability of stream process execution. We formally describe different levels of reliability for DSM, like *lossless*, *delay-limited* and *intra-order preserving* reliability level (see Section 5.1). We have chosen to focus on this reliability levels because we state that loss of data stream elements is in general not tolerable by various DSM applications (e.g., telemonitoring in healthcare). Moreover, also the delay of data stream elements has to be limited in order to be able to react to certain conditions. In particular for critical health conditions, the corresponding data stream elements should be available within limited and predictable time. Lastly, also the order of the data stream elements within a data stream (intra-order) has to be conserved.

Secondly, the proposed efficient reliability algorithm for DSM presented in this chapter is designed to guarantee these reliability levels. Generally, we can distinguish two groups of reliability techniques. First, *hot-standby strategies* impose that DSM processing is performed in parallel at two different nodes in the network. We state that these strategies are too resource demanding for the presented application areas. Second, *passive-standby strategies* do not process in parallel but regular checkpoint messages allow a passive standby-node to seamlessly take over processing in case failures. We state that, passive-standby is more appropriate with respect to the presented applications. Moreover with regard to passive-standby, we state that due to

resource limitations the complete migration of all DSM processing workload that was performed at the failed node to a single standby-node is not always possible. In particular, since the goal is to make efficient use of available resources, the infrastructure has to deal with reliability at a more *fine grained level* of data stream operators. For example, if a patient's PDA fails, which has previously hosted three operators, the most efficient failure handling can be that two operators are migrated to the patient's base station and one operator is migrated to the patient's smartphone. Obviously, the *operator migration* in case of failures has to be seamless, which means that no streaming data is getting lost. In order to achieve this the infrastructure has to apply efficient and reliable operator checkpointing algorithms during the runtime of operators.

5.1 Reliability Levels of Stream Process Execution

The presented model in Chapter 4 describes all interactions of the DSMS with the outside world through outside data streams (o*w*-edges). Therefore from the outside world point of view, the internals of DSMS are seen as a black box. Based on this fact, we can define the reliability level of a DSMS faced by an outside user by comparison of a *real-world DSMS* to an *ideal DSMS*. The *ideal DSMS* system is a virtual system which is executing all stream processes in a proper way according to the stream process definition (SPD). This means that no delays are happening during stream process execution, no data stream elements get lost, and no wrong data stream elements are produced due to failures. In contrast the *real-world DSMS* is an error-prone DSM system that has to deal with failures happening in the real-world.

In the following, we compare the *output outside data streams* $\mathrm{DS}_{\mathrm{ideal}}$ of an ideal DSMS with output outside data streams $\mathrm{DS}_{\mathrm{real}}$ of a real-world DSMS. Since the ideal and real DSMS are fed with exactly the same input outside data streams generated by the outside world, we only need to compare output outside data streams.

The highest level of reliability is achieved by a real world system with identical output data streams as the ideal system. In this case, there is no difference to the ideal DSMS from the outside world's point of view.

Definition 5.1 *Ideal reliability level* is defined as having for all output data streams: $\mathrm{DS}_{\mathrm{ideal}} = \mathrm{DS}_{\mathrm{real}}$ ◇

Unfortunately, real world systems have to cope with failures and therefore may not produce the same result as ideal systems. Real world systems may sacrifice this ideal reliability also on purpose in order to use available

resources more efficiently. Since not all applications have the same requirements on reliability it is beneficial to have different levels of reliability applied to different stream process definitions. In the following, we consider reliability levels to be individually defined for different stream processes.

For this reason, we define the following two subset data streams of a real world output outside data stream.

Definition 5.2 The *correct data stream* DS_{corr} contains the subset of the data stream elements appearing in the real-world data stream DS_{real} which have the same (correct) sequence number ξ and payload information pd as data stream elements appearing in the corresponding ideal data stream DS_{ideal}:

$$DS_{corr} \subset DS_{real}$$
$$\forall de_{corr} \in DS_{corr}(\exists! de_{ideal} \in DS_{ideal} :$$
$$(de_{ideal}.\xi = de_{corr}.\xi \wedge de_{ideal}.pd = de_{corr}.pd))$$

$\diamond$

Elements in the correct data stream may have a different global timestamp τ compared to the *corresponding* data stream element in the ideal data stream because they may be delayed during processing. Correspondence is defined on the basis of the sequence number ξ and payload information pd.

Definition 5.3 The *incorrect data stream* DS_{fail} contains the subset of the data stream elements appearing in the real-world data stream DS_{real} which have no corresponding data stream elements appearing in the ideal data stream DS_{ideal}. Correspondence is defined on the basis of the sequence number ξ and payload information pd:

$$DS_{fail} \subset DS_{real}$$
$$\forall de_{fail} \in DS_{fail}(\nexists de_{ideal} \in DS_{ideal} :$$
$$(de_{ideal}.\xi = de_{fail}.\xi \wedge de_{ideal}.pd = de_{fail}.pd))$$

$\diamond$

Furthermore, we are able to degrade the ideal reliability level in three orthogonal dimensions considering *loss, delay,* and *order*. In this thesis, we have intentionally *not* modeled *accuracy* of data stream elements as reliability criteria as in [C+02, BBMS05]. The reason for this is, we state that in several application domains (e.g., for healthcare applications) inaccurate DSM processing is generally not tolerable or very application specific. In this thesis, we consider a binary approach where a data stream element is either consid-

ered as correct or incorrect. Inaccurate (incorrect) data stream elements are considered as invalid and therefore result in loss of data stream elements.

Loss is defined as having less correct data stream elements DS_{corr} in the real-world output outside data stream than the ideal output outside data stream, but may have additional incorrect data stream elements DS_{fail}. The loss definition allows for processing delays of data stream elements (a different global timestamp τ). For this reason, loss is independent or orthogonal to reliability levels constraining the delay and order of data stream elements.

Definition 5.4 *Limited-loss reliability level* (LILO) is defined as having:

$$\begin{aligned} DS_{real} &= DS_{corr} \cup DS_{fail} \\ |DS_{ideal}| \geq |DS_{corr}| &\geq |DS_{ideal}| * LF \quad LF \in (0,1] \\ |DS_{fail}| &\leq |DS_{ideal}| * EF \quad EF \in [0,\infty) \end{aligned}$$

where LF is a maximum allowed loss factor and EF is a maximum allowed error factor. ◇

Definition 5.5 *Lossless reliability level* (LOLE) is a special case of the loss reliability level, where $LF = 1$ and $EF = 0$. ◇

Revisiting our example scenario of Section 2.2, patient Fred will require a lossless reliability level if the abnormalities in his physiological signals appear very infrequent but still have severe consequences for his health and therefore for the adaption of treatment. This situation arises commonly with cardiac arrhythmias like arterial fibrillation. This disturbances occur maybe only once a week for a short period of time. If data stream elements that indicate an arrhythmia of Fred's heart are lost this has severe consequences for the correct treatment of Fred.

Delay is defined as having for correct data stream elements in the real-world outside data stream a certain delay in the global timestamp τ compared to DS_{ideal}. Delay is not putting any constraints on missing data stream elements compared to the ideal data stream, which are subject of loss. In general, delay is putting no constraints on the order of data stream elements within global time.

Definition 5.6 *Limited-delay reliability level* (LIDE) is defined as having:

$$\begin{aligned} \forall de_{corr} \in DS_{corr} (\exists! de_{ideal} \in DS_{ideal} : \\ de_{ideal}.\xi = de_{corr}.\xi \wedge de_{ideal}.pd = de_{corr}.pd \\ \wedge de_{ideal}.\tau \leq de_{corr}.\tau \leq de_{ideal}.\tau + \triangle\tau) \end{aligned}$$

where $\triangle\tau$ is a maximum allowed delay. ◇

Definition 5.7 *Delay-free reliability level* (DEFR) is a special case of the limited-delay reliability level, where $\triangle\tau = 0$. ◇

Notice that in the delay-free case, the order of the data stream elements is preserved. A detailed explanation is given in the proof of Lemma 5.13.

Revisiting our example scenario of Section 2.2, patient Fred will tolerate a limited-delay reliability level. Limited-delay reliability guarantees that his data streams are processed within a certain time. Delay times in the order of seconds in stream processing are quite short to the normal time of intervention needed to apply the telemonitoring results. Even in case of an severe emergency, e.g., heart attack, the ambulance needs at least multiple minutes to come. Therefore, some seconds due to delays on stream processing are negligible.

Order is defined for correct data stream elements in the real-world outside data stream and is coming in two flavors. One is *intra stream order*, which means that the ordering of the data stream (according to the global timestamp τ) is preserved. Order considers only the available correct data stream elements and therefore does not allow for loss. Order also tolerates processing delays as long as the order is not messed up.

Definition 5.8 *Intra stream order preserving reliability level*
(IASO) guarantees:

$$\forall de_{corr}(\xi) \in DS_{corr} : de_{corr}(\xi).\tau \leq de_{corr}(\xi + 1).\tau$$

◇

The other aspect is *inter stream order*, for which we compare additionally if the temporal-order of data stream elements is preserved between data streams of the same stream process. To be more precise, the relative order between every pair of the correct data stream elements of two data streams DS_{corr1} and DS_{corr2} has to be maintained.

Revisiting our example scenario of Section 2.2, we consider a stream process monitoring our patient Fred's that is producing two output streams. One output stream is the average blood pressure of the last minute and the other is the average heart rate of the last minute. The ideal system always produces the data stream element with the blood pressure average of the last minute before the data stream element with the heart rate average. In this case, the inter stream order reliability level is guaranteeing this behavior also for the real-world system.

Definition 5.9 *Inter stream order preserving reliability level*
(IESO) guarantees:

$$\forall de_{corr1} \in DS_{corr1}(\forall de_{corr2} \in DS_{corr2} :$$
$$((de_{corr1}.\xi \leq de_{corr2}.\xi) \wedge (de_{corr1}.\tau \leq de_{corr2}.\tau))$$
$$\vee((de_{corr1}.\xi \geq de_{corr2}.\xi) \wedge (de_{corr1}.\tau \geq de_{corr2}.\tau)))$$

◇

In general, patient Fred will be able tolerate disorder of data stream elements. A reason for this is that the DSMS is always able to detect disorder within data streams and reorder them appropriate at network level to guarantee the intra stream order reliability level. For inter stream order reliability level, Fred's data stream analysis operators are able to detect disorder at application level. For example, Fred's heart rate data stream is arriving in incorrect order compared to Fred's blood pressure data stream at a common analysis join operator. In this case, additional time information about the generation of a data element at sensor nodes can be used to verify time synchronization at application level. Further details on generation time are presented in Section 4.1.2. Of course, we assume that sensors are working on synchronized clocks.

Figure 5.1 illustrates the reliability levels of DSM and their relationship. The three levels of reliability LILO, LIDE, IASO are orthogonal sets. Lossless (LOLE), Delay-free (DEFR), and inter stream order (IESO) are subsets of LILO, LIDE, and IASO respectively. The only exception of orthogonality is between DEFR and IESO/IASO are related because delay-free reliability guarantees inter-stream and intra-stream order reliability but not vice versa. Formally, reliability levels are sets of DSM systems. In contrast, a real-world DSMS is an element working at certain reliability levels. Fig. 5.1 illustrates the relationship between these levels. The ideal DSMS which is working at ideal reliability level and therefore fulfilling all other reliability levels is shown in the center of Fig. 5.1. If a DSM system is not fulfilling a certain reliability level, the system is not part of the reliability level set. In this case, the actual reliability of the system with respect to the investigated reliability level (loss, delay, order) is not defined and therefore unpredictable.

Lemma 5.10 All reliability levels have a common intersection. ◇

Proof. The common intersection is proven because the ideal DSMS element fulfills all defined reliability levels. This is given by the definitions of the different reliability levels which define degradations compared to the result of an ideal DSMS. □

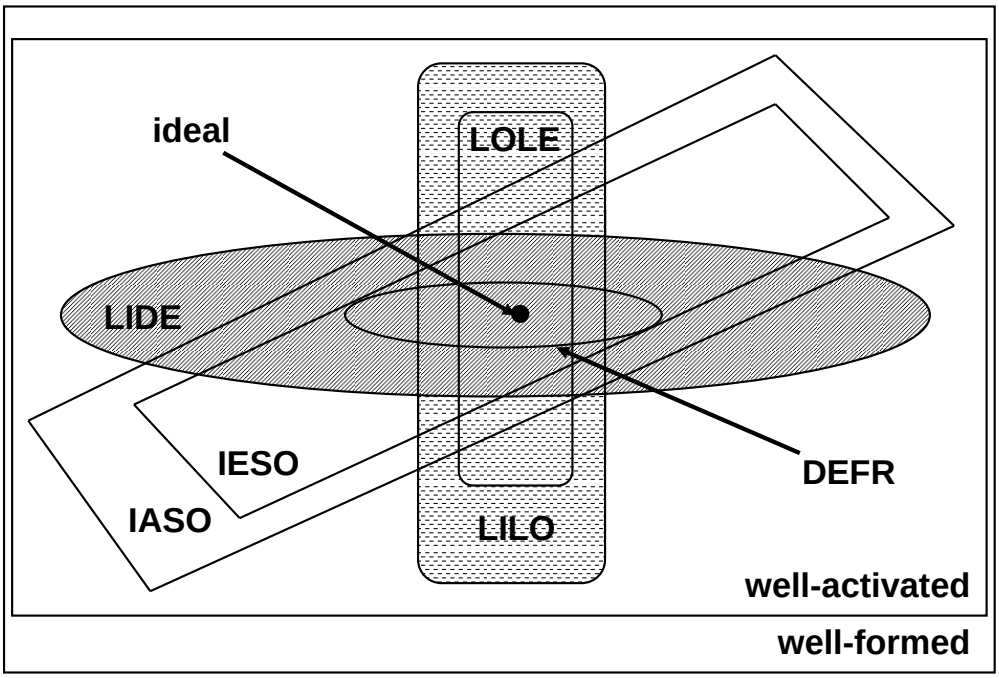


Figure 5.1: Relationship between Reliability Levels of DSM

Lemma 5.11 The more restrictive reliability levels LOLE, DEFR, IESO are subsets of the corresponding reliability levels LILO, LIDE, IASO.
$\mathrm{LOLE} \subseteq \mathrm{LILO}$
$\mathrm{DEFR} \subseteq \mathrm{LIDE}$
$\mathrm{IESO} \subseteq \mathrm{IASO}$
$\diamond$

Proof. Each lossless (LOLE) DSMS fulfills also the criteria for limited-loss (LILO) reliability level where the loss factor $\mathrm{LF} = 1$.

Each delay-free (DEFR) DSMS fulfills also the criteria for limited-delay (LIDE) reliability level where the maximum allowed delay $\triangle\tau = 0$.

Each inter-stream order (IESO) preserving DSMS fulfills also the criteria for intra-stream order preserving (IASO) reliability where the pair of compared data streams points to the same data stream instance $\mathrm{DS}_{\mathrm{corr1}} = \mathrm{DS}_{\mathrm{corr2}}$. $\square$

Lemma 5.12 The three reliability levels LIDO, LIDE, IASO are orthogonal sets. Orthogonality describes in this case, that the sets are independent and not subsets of each other. Because of Lemma 5.10 there is a common intersections between the sets. ◇

Proof. In order to proof this lemma, we have to show that none of these three sets (LIDO, LIDE, IASO) are subsets of each other.

LIDE $\not\subseteq$ LILO: There are DSM systems, that produce data stream elements with limited delay (LIDE) but without any guarantee with respect to the amount of loss (LILO) of data stream elements.

LILO $\not\subseteq$ LIDE: There are DSM systems, that produce data stream elements with limited loss (LILO) but without any guarantee with respect to the amount of delay (LIDE) of data stream elements.

LILO $\not\subseteq$ IASO: There are DSM systems, that produce data stream elements with limited loss (LILO) but without any guarantee with respect to preserving the order (IASO) of data stream elements.

IASO $\not\subseteq$ LILO: There are DSM systems, that produce data stream elements with correct order (IASO) but without any guarantee with respect to the amount of loss (LILO) of data stream elements.

IASO $\not\subseteq$ LIDE: There are DSM systems, that produce data stream elements with correct order (IASO) but without any guarantee with respect to the amount of delay (LIDE) of data stream elements.

LIDA $\not\subseteq$ IASO: There are DSM systems, that produce data stream elements with limited delay (LIDE) but without any guarantee with respect to preserving the order (IASO) of data stream elements. □

Lemma 5.13 The delay-free (DEFR) reliability level is also a subset of the inter-stream order preserving reliability level.
DEFR $\subseteq$ IESO
◇

Proof. Each DSMS that guarantees delay-free (DEFR) data stream processing also preserves the inter-stream order of the produced data stream elements. This is caused by the fact that the order of data stream elements within a data stream can only be disturbed by additional delays. In the case of DEFR, there is no delay of data stream elements as in the ideal case. For this reason, the order of the correct produced data stream elements is also correct. Still, there is possible loss of data stream elements and therefore DEFR is independent with respect to the limited-loss (LILO) reliability level. □

Finally, as an important prerequisite for guaranteeing the different reliability levels all executed stream processes have to be well-activated and their corresponding stream process definitions have to be well-formed.

5.2 Failure Model

In the following, we describe a failure model based on the given DSMS model in Chaper 4.

Definition 5.14 *Failure of an operator instance* o_{fail}: This failure affects the stream process SP_{fail} which is using the affected operator:

$$SP_{fail} \in \widehat{SP}|_{o_{fail} \in SP_{fail}.OS}$$

Moreover, within SP_{fail} besides o_{fail} all data streams DS_{fail} from DSS are affected that are connected to a port of o_{fail}:

$$DS_{fail} \in SP_{fail}.DSS|_{(DS_{fail}.ip \in o_{fail}.\nu.ot.IP) \vee (DS_{fail}.op \in o_{fail}.\nu.ot.IP)}$$

◇

Definition 5.15 *Failure of a data stream* DS_{fail}: This failure affects the stream process SP_{fail} which is using the affected data stream:

$$SP_{fail} \in \widehat{SP}|_{DS_{fail} \in SP_{fail}.DSS}$$

Moreover, within SP_{fail} besides DS_{fail} all operator instances o_{fail} that are connected by DS_{fail} are affected:

$$o_{fail} \in SP_{fail}.OS|_{(DS_{fail}.ip \in o_{fail}.\nu.ot.IP) \vee (DS_{fail}.op \in o_{fail}.\nu.ot.OP))}$$

◇

Definition 5.16 *Failure of a host* h_{fail}: This failure affects all stream processes SP_{fail} having running operator instances, which are hosted by the affected host:

$$SP_{fail} \in \widehat{SP}|_{(\exists o_{fail} \in SP_{fail}.OS : o_{fail}.h = h_{fail})}$$

Within each affected stream process SP_{fail} the following operators o_{fail} are affected:

$$o_{fail} \in SP_{fail}.OS|_{o_{fail}.h = h_{fail}}$$

Moreover, also the following data streams DS_{fail} connecting to an affected operator o_{fail} are affected by the failure:

$$DS_{fail} \in SP_{fail}.DSS | (DS_{fail}.ip \in o_{fail}.\nu.ot.IP) \vee (DS_{fail}.op \in o_{fail}.\nu.ot.IP)$$

$\diamond$

Our failure model assumes all failures to be *fail-stop failures*. Fail-stop failures are failures where the affected part is completely stopping its work. Other (non-fail-stop) failures may result in improper results, e.g., an operator instance is producing wrong output data stream elements or an data stream is suffering from distortion and therefore modify payload information during transfer. In practice, the later failure cases can be avoided by having proper means, e.g., cyclic redundancy checks, within the DSMS infrastructure to detect transmission errors or faulty operators. The outside world system is assumed to be always working correct as discussed in Section 4.2.

5.3 States within Stream Process Execution

During runtime of a stream process in a DSMS different kinds of states are generated:

- **Operator state** ($s(\tau)$). This is the most obvious state and has already been defined in Section 4.1. This state is generated by each running operator instance during the processing of data streams.

- **Time context** ($TC(\tau)$). Each operator instance has to know its current stream-time context.

  $$TC(\tau) = \left\langle \Xi^{in}, \Xi^{out} \right\rangle$$

  Ξ^{in}, Ξ^{out} refer to the stream-time of last processed input and output data stream elements (c.f. operator definition in Sec. 4.1).

- **Transfer state** ($TS(\tau)$). In a real world system the processing and transmission of a data stream DS is suffering from delays. Therefore, there may be at a given global timestamp τ some data stream elements in a state of transfer along an edge $e = \langle x, op, y, ip, \triangle\tau \rangle$ of a stream process. The state of the data stream between op and ip is called *transfer state* TS. The transfer state along an edge is given as subset of DS:

  $$TS(\tau) = DS(\xi_s, \xi_e)$$

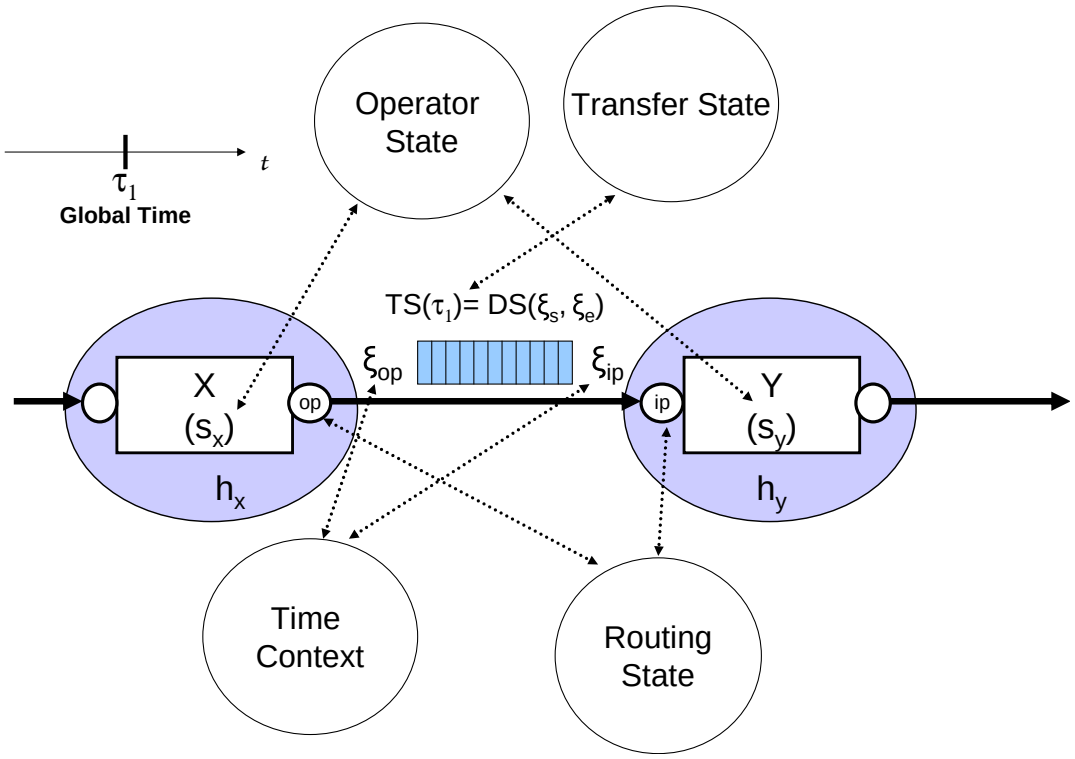


Figure 5.2: States of a Stream Process

where $\xi_s \leq \xi_e$ and $|DS(\xi_s, \xi_e)| = \xi_e - \xi_s$ is the number of elements in the transfer state. ξ_s refers to the element with oldest stream-time and ξ_e refers to the element with the newest stream-time in state of transfer. In practice, an *acknowledgment* protocol between consumers and producer of a data stream will describe elements in transfer as *non-acknowledged* elements.

- **Routing information** ($RI(\tau)$). Finally, for each edge the stream process execution has to know the host which is currently hosting the source vertex x and the host of the destination vertex y, where the data stream has to be sent to. This *routing information* h_x, h_y is also considered as state of the stream process.

Figure 5.2 illustrates the states within a stream process.

5.4 Consistency Within a Stream Process

Based upon the previous definitions of states within stream processes, we are able to define consistency constraints for relations between the *time context* of the *operator state* and *transfer state*.

Exact consistency ensures that at every global timestamp τ all operator instances are working consistently to the transfer states of the data streams in between. For this reason, also all operator instances are working consistent to each other. In this case, exact consistency is guaranteeing a lossless reliability level, because loss would cause an inconsistency.

Definition 5.17 *Exact Consistency*, is defined by the following: For all pairs of two operators x, y which are connected via an edge e from $x.op$ to $y.ip$ at any point in global time τ the following has to hold

$$\begin{aligned} x(\tau) &= \left\langle OT_x, \Xi_x^{in}, \Xi_x^{out}, s_x, h_x, \tau \right\rangle \\ y(\tau) &= \left\langle OT_y, \Xi_y^{in}, \Xi_y^{out}, s_y, h_y, \tau \right\rangle \\ TS(\tau) &= DS(\xi_s, \xi_e) \\ (\xi_{op} = \xi_e) &\wedge (\xi_{ip} = \xi_s) \end{aligned}$$

where ξ_{op} is current stream time of the last element produced at the output port of x taken from Ξ_x^{out} and ξ_{ip} is the current stream time of the last element consumed at the input port of y taken from Ξ_y^{in}. ◇

Based on the deterministic operator model we are able to relax the exact consistency constraint. Relaxed consistency allows for the sender x to resend data stream elements if necessary and for the receiver y to receive data stream elements again. For internal vertices within the stream process this assumptions are valid, because an appropriate transport mechanism within the DSMS is able to offer this. For interaction with the outside world along o*w*-edges this assumptions are also valid as stated in Section 4.2.

Definition 5.18 *Relaxed Consistency*, is defined by the following: For all pairs of two operators x, y which are connected via an edge e from $x.op$ to $y.ip$ at any point in global time τ:

$$\begin{aligned} x(\tau) &= \left\langle OT_x, \Xi_x^{in}, \Xi_x^{out}, s_x, h_x, \tau \right\rangle \\ y(\tau) &= \left\langle OT_y, \Xi_y^{in}, \Xi_y^{out}, s_y, h_y, \tau \right\rangle \\ TS(\tau) &= DS(\xi_s, \xi_e) \\ (\xi_{op} \leq \xi_e) &\wedge (\xi_{ip} \geq \xi_s) \end{aligned}$$

◇

Revisiting our example scenario of Section 2.2, patient Fred's heart activity and blood pressure is monitored with the example process in Fig. 3.4. In this stream process the *ECG Acquisition* operator is sending data stream elements containing samples of Fred's current heart activity to the *ECG Variability* operator which is processing the variability within Fred's heart activity. Having exact consistency enforced, the *ECG Acquisition* operator is allowed to send each data stream element *only once*. This means that the newest element in transfer has exact the stream timestamp of the last element sent. Moreover, exact consistency forbids that elements get duplicated during transfer. This means that the oldest element in transfer is exactly the next data stream element the *ECG Variability* operator is expecting.

When relaxing consistency, we allow for the *ECG Acquisition* operator to *re-send* data stream elements. This results in having duplicate heart activity data stream elements within transfer state. The stream timestamp of the last element sent is allowed to be smaller or equal than the newest element in transfer state. Relaxed consistency allows also for data stream elements to exist duplicated in transfer state. This means that the oldest heart activity sample in transfer has to have a smaller or equal stream time (sequence number) than the next data stream element expected by the *ECG Variability* operator.

Lemma 5.19 Having relaxed consistency enforced at all times during the processing of a DSMS guarantees lossless reliability. ◇

Proof. Relaxed consistency from Def. 5.18 is defined by $(\xi_{\mathrm{op}} \leq \xi_e) \wedge (\xi_{\mathrm{ip}} \geq \xi_s)$. Compared to exact consistency from Def. 5.17 where $(\xi_{\mathrm{op}} = \xi_e) \wedge (\xi_{\mathrm{ip}} = \xi_s)$ there are additional elements allowed between ξ_{op} and ξ_e and between ξ_s and ξ_{ip}. These elements are additional older elements and because of the deterministic finite state machine model of the operator this elements must be duplicates of the elements which caused the current state. Therefore, the elements can be safely discarded without affecting the current or future state of the DSMS. In this case, relaxed consistency *is also* guaranteeing a lossless reliability level. Also with respect to outside world interactions (see Section 4.2) relaxed consistency is applicable. The outside sender assumption is stating the input data streams from outside word can be reread and the outside receiver assumption allows for re-sending data stream elements to the outside world. □

5.5 Distinction Between Delays and Failures

In a real-world DSM system the processing and transmission of data stream elements is subject of delays. Our model has inherently accepted delays as part of operator types and stream process definitions where vertices and edges have maximum allowed delay constraints given. Given this delays, we can define *temporary failures*, where the effect of a failure is only temporary, e.g., a wireless network disturbance. The term *temporary* indicates that no delay constraints within the stream process definition are violated. On the other hand, if a failure is persistent in a way that the delay constraints are exceeded we consider the failure as *permanent failure*. In a real-world implementation of DSMS the temporary failures are usually compensated for through the use of buffers in between the operators. Contrarily, permanent failures have to be treated in a more sophisticated way. In order to follow the delay constraints even in case of failures the real-world DSMS infrastructure has to apply an *efficient reliability strategy* to handle the failure before delay constraints are exceeded. Since the reliability strategy needs some time τ_r to recover from the failure situation, the DSMS has to start failure handling when the current delay exceeds the maximum allowed delay $\triangle\ \tau$ minus the recovery time τ_r. Fig. 5.3 illustrates the temporal behavior of an example data stream during a failure and recovery of a DSMS. The illustration uses a two-axis visualization of the data stream. The x-axis is the stream-time or sequence number and the y-axis shows the processing time of the data stream element. In order to measure delays in processing time, the illustration contains the ideal and real world data stream elements. Shortly after processing of the third input data stream element, the DSMS has a failure. Before exceeding the maximum allowed delay, the failure becomes a permanent failure and the DSMS starts recovery. At this point in global time, the ideal DSMS would already have processed the seventh data stream element (see thin horizontal line in Fig. 5.3). Obviously these data stream elements have to be stored in appropriate queues in the real-world case, either directly within the sensors or at an input interface for sensor devices. After the recovery time τ_r, the fourth data stream element is processed by the real world DSMS while data stream elements 5,6,7, and 8 have queued up and producing a congestion. Like other time-constrained systems, a real-world DSM system is not able to be execute at full CPU utilization [LL73]. For this reason, there are unused CPU resources available that are able to work off the congestion during the catchup time τ_c. This assumption includes the fact that also mobile device have spare CPU cycles for such cases. After τ_c the congestion is worked off and the average delay of an incoming data stream element is the same as before the failure. Using the reliability levels introduced in Section 5.1, the illustrated

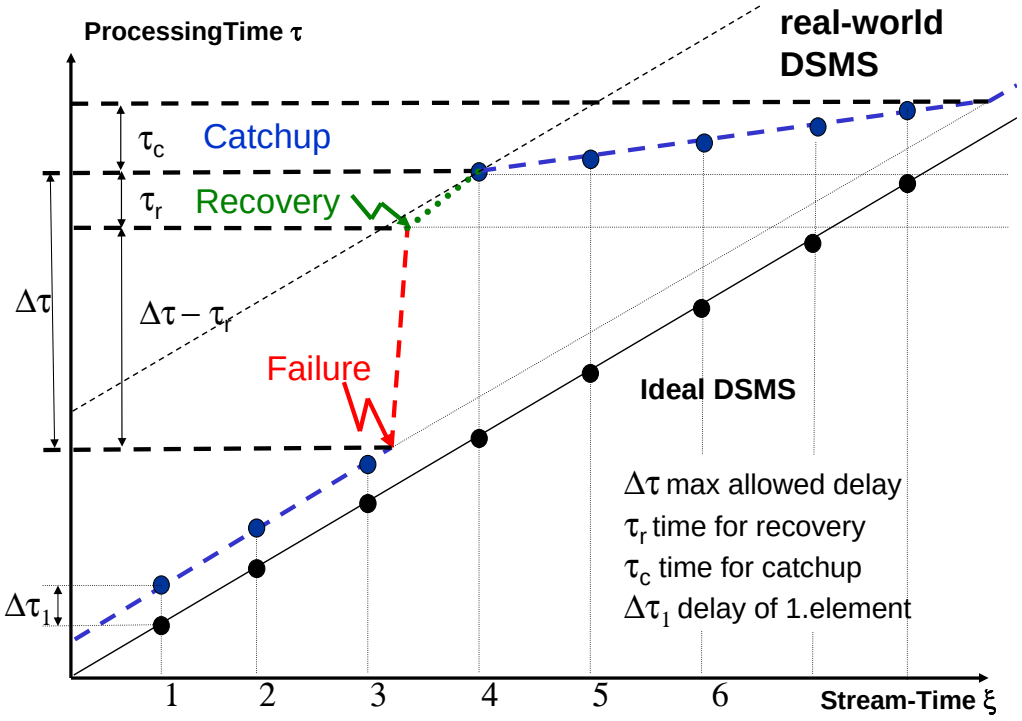


Figure 5.3: Temporal Behavior of a Failure

real-world data stream fulfills lossless reliability, is intra-order preserving, and has limited-delay reliability. Of course, the ideal data stream fulfills ideal reliability.

5.6 Failure Handling of the DSM Infrastructure

In general, the DSMS is able to deal with failures in two different ways. Firstly, *transparent failure handling* is completely done by the DSM infrastructure without any user intervention or interaction with the application. Secondly, if transparent failure handling is not possible, the DSMS can apply *application-defined failure handling*.

Application-defined failure handling can be applied in two flavors. Firstly, *alternative processing branches* are defined in the stream process definition and allow modifications for different branches within a stream process. For

example, this allows to continue data stream processing even in the case no other provider in the network is able to host the failed operator instance for recovery. This can be done by executing an alternative processing branch given in the stream process definition which is bypassing any operator type that is not available. In this thesis, we present this failure handling approach for reason of completeness but we have not evaluated the outcome of this approach in detail. Moreover, there are some issues on alternative processing branches that are presented in the outlook of thesis in Chapter 9.

Secondly, *invocation of traditional processes* may handle failure situations where the proper execution of the stream process is not possible. These traditional processes are defined by the process designer (e.g., the caregiver) and they describe how to cope with the failure. For example, patient Fred is leaving the house and therefore causing an network connection failure on the wireless connection between his PDA and his base station at home. An appropriate alarm process can inform Fred, e.g., by sending him an SMS to his cellular phone, that he is currently not monitored because of network disconnection and returning back in connection range of his monitoring system will solve the issue.

The first-class solution proposed and evaluated in this thesis is to handle failures *transparently* by the DSM infrastructure. The remainder of this chapter will emphasize on transparent failure handling. Transparent failures are further distinguished between *temporary* and *permanent* failures (see Section 5.5).

Temporary failures, e.g., a temporary network disconnection (loss of messages) or a temporary failure of a provider which is able to recover within the maximum allowed delay time $\triangle\tau$, are compensated by the transfer state which is kept in output buffers of upstream providers. For recovery, the upstream provider is able to re-send the buffered data stream elements. In order to prevent from long delays and huge buffer sizes, a temporary failure becomes a permanent failure before exceeding the maximum allowed delay $\triangle\tau$.

Permanent failures, e.g., a permanent network disconnection or failure of a provider, require to *migrate* an operator instance with its aggregated operator state from the affected provider to another suitable provider. *Operator migration* implies the continuation of an operator instance from a recent checkpoint on a new provider in order to allow for seamless continuation of DSM, and eventually the stopping of an old running operator instance. Ensuring consistency between operator states even in case of failures is a crucial constraint for seamless continuation of data stream processing. Details on handling of permanent failures are described in the remainder of this chapter.

Consequences of a failure in distributed DSMS usually affect more than one node of the infrastructure because communicating operator instances of a stream process may be executed at different nodes. Therefore, it is vital for proper failure handling of a DSMS that all affected nodes detect the failure or have to be informed about the failure. Therefore, failure detection is inevitable for proper failure handling.

In this thesis, we do not focus on the detection of failure situations in a DSMS. We have implemented some basic failure detection mechanism based on acknowledgement and heartbeat messages that are exchanged between communicating hosts within the DSM infrastructure. Leveraging the nature of stream processing, combination of transmission of data streams and observation of operator instances is reasonable. The DSM infrastructure offers a reliable FIFO-transport for DSM messages, which leverages an acknowledge protocol as described in [TS01]. This FIFO-transport already guarantees intra-order reliability of DSM as defined in Section 5.1. The acknowledge protocol is also used to detect failures along data stream connections between operator instances, which are the edges in the stream process definition. This allows providers of connected operator instances to pairwise observe each other. Detected acknowledge failures are indicating a failure along the connecting edge but the provider cannot distinguish between a network failure, a failure of the other provider, or a just the failure of the operator instance (according to the three kinds of failures defined in the failure model of Section 5.2). The following failures are currently detectable by the proposed DSMS infrastructure:

Operator failure. In this case, the operator instance has failed but the provider node is still up and running.

Host failure. In this case, a host of the DSMS has failed and therefore all running operator instances at the host have failed consequently.

Host overload. In this case, a host of the DSMS has not failed but due to high workload the delay constraints are not maintainable. A typical task of a DSM infrastructure is to monitor the computation load of each provider and publishes this information via global repositories (see Chapter 3). This load information is replicated throughout the DSM infrastructure and allows for balancing the load within the overall system, i.e., by allowing to locally choose the least loaded provider offering a particular operator.

Network disconnection. In this case, all hosts behind the network disconnection are unreachable and therefore affected from the failure. Because of this, all operator instances running on the affected hosts have to be migrated to unaffected hosts.

In particular, detection and handling of network disconnections which separates the DSM infrastructure into disconnected partitions is not trivial and

out of scope of this thesis. There is a plethora of work in the field of networking research dealing with failure detection [KSS04, Sri06, HWS04]. In this work, we assume the DSM infrastructure is able to detect the failure situation and there are significant numbers of unaffected hosts in the network available to take over the workload and compensate the failure.

5.7 Operator Migration

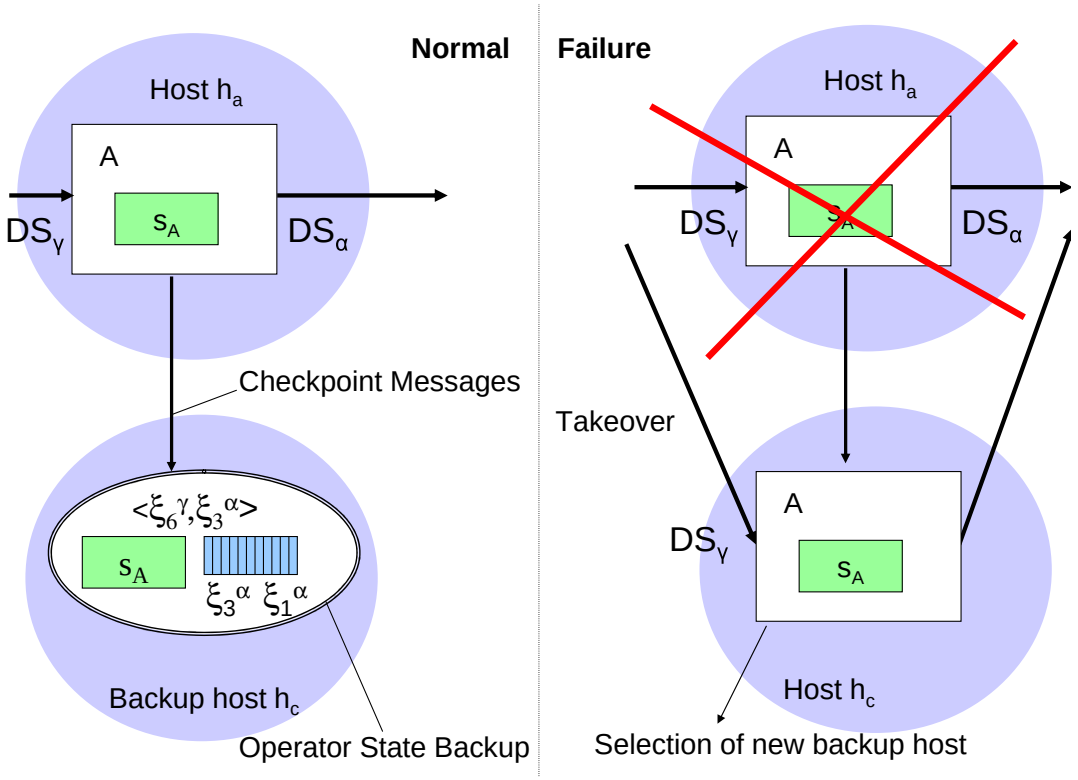


Figure 5.4: Operator Migration

During runtime of a DSMS, failures as described in Section 5.2 are likely to happen. If a failure situation persists longer than $\triangle\tau - \tau_r$ (see Figure 5.3) at any edge or vertex of a running stream process the failure becomes a permanent failure. In this case, the DSM system applies transparent failure handling at first. For transparent failure handling, the DSMS has actively apply some mechanisms in order to recover from the failure situation and keep the

required reliability levels. With respect to the applications with high reliability demands, we focus on lossless and limited-delay reliability. The expected failure situations within our model are operator failure, network disconnection, and host failure. Whereas some kinds of operator failures or data stream failures may be solved by restarting an operator at the same host or reestablishing a data stream between to operators again, we most failures require that running operator instances have to be migrated from affected hosts to other unaffected hosts. This movement of an operator instance from one host to another is called *operator migration*. Of course, operator migration also involves to redirect the edges (data streams) between them. Figure 5.4 illustrates how a failure can be handled by using to operator migration.

In order to allow for operator migration in case of failures, the DSM system needs access to a recent state of the affected operator instances. For this reason, we apply a passive-standby approach based on checkpointing [CL85, HBR+05].

5.8 Operator Checkpointing

In our model, we define an *operator checkpoint* as the reliable storage of the current state of the operator instance and the transfer state of all streams produced by this operator instance at a reliable *backup host*. The checkpoint is passed to the backup host via sending a checkpoint message. By applying an acknowledgement on this message the backup provider guarantees the reliable storage of the checkpoint. The selection of the backup host is done by the host of the corresponding operator instance and based on locally replicated load information. Furthermore, we assume the backup host does not fail. Otherwise, multiple backup hosts are needed to cope with such failures. In a DSMS, the backup host of an operator checkpoint is ideally able to be the host of the corresponding operator. In case of a failure, the backup host, which has the operator checkpoint locally available, is the destination of operator migration.

An important prerequisite of operator migration is to ensure the availability of a recent operator state backup at the backup node in the DSM infrastructure. In order to achieve consistent recovery, which means that operator migration will not lead to loss of data, the DSMS applies an efficient operator checkpointing strategy during runtime of the stream process.

An operator checkpoint subsumes the current state of an running operator instance including the transfer states of the outgoing data streams. These states (as introduced in Sec. 5.3) are categorized by size and frequency of changes in Table 5.1. We consider the operator state (s) as changing fre-

quently because every incoming data stream element may trigger a state change. The size of the operator state is according to a finite state model usually constant and compared to other states it is considered as medium. The time context (TC) consists of a stream timestamp for each incoming and outgoing data stream therefore the size is constantly small but changes frequently. The transfer state (TS) is considered as varying in size and rather medium to big. We assume the size transfer state is usually bigger than the size of the operator state. Finally, the routing information (RI) is giving the destination providers for each outgoing and incoming data stream. This information is constant and small in size. The frequency of change is rather low compared to the other states because changes in routing information are happening only when operator instances migrated to other providers.

State	Changes	Size
Operator State (s)	frequent	constant medium
Time Context (TC)	frequent	constant small
Transfer State (TS)	frequent	varying medium - big
Routing Information (RI)	infrequent	constant small

Table 5.1: Categorization of States

In our model, the node hosting the producing operator instance is responsible for keeping the transfer states. Moving this responsibility to the consumer side would cause unnecessary overhead since our DSM models assumes a *multiple consumer - single producer* pattern for data streams. This means that, a single output port of an operator is able to feed multiple data stream that are connected to input ports of *downstream operators*. Downstream operators are operator instances which are processing data stream elements that are produced by the regarded operator. Similarly, *upstream operators* are operator instances that produce data stream elements processed by the regarded operator.

In order to reduce the effort for operator checkpointing at backup nodes, we introduce the following additional assumption for the reliability strategies. Only the most recent checkpoint is kept at the backup node. Therefore when a host is sending a newer checkpoint the older one is always overwritten: For this reason it is not possible for the DSMS to go back further than to the most recent checkpoint stored at the backup node during operator migration.

5.8.1 Consistency Requirements on Checkpointing

Furthermore, the reliability strategy of the DSMS has to guarantee *relaxed consistency* (see Section 5.4) in order that the DSMS can guarantee *lossless reliability level* to the application. This reliability level has to be guaranteed

even in case of multiple failures for which one ore more operators are migrated and have to be recovered from their last checkpoints. At the same time, all other operators not affected from failures are keeping their current state.

In our investigations so far, the transfer state is part of the checkpoint messages and used to backup data stream elements for consuming operators. This allows data stream elements in transfer state to be re-sent in case of failures. Conceptually, there is one transfer state for each output port of a running operator instance keeping data stream elements which are in transport and have not yet been processed by the consuming operator instances. As mentioned previously in this sections, each output port is able to feed multiple input port and therefore multiple operator instance. Still, the multiple consumers share one transfer state, which is keeping data stream elements. Whenever, consumers no more *rely* on the transferred data stream elements an appropriate *acknowledgement (ACK)* message is sent to the producing output port. This message contains a stream timestamp ξ_i^{ack} which indicates the timestamp of last data stream element the consuming operator as processed of output port i. All data stream elements that have been acknowledged by all consuming operator instances of output port i are subject to *transfer state trimming*.

Considering a failure, the migrated operator instance restart from the previous checkpoint and is able to retrieve all data stream elements kept in transfer states of upstream operators again.

In Figure 5.5, we present a *single operator failure scenario*. In this case, we assume the execution of a stream process with a sensor (outside world) and two subsequent operators in the data flow (see lower right corner of Fig. 5.5). Operator checkpointing is applied by the DSMS for the operators A and B. In case operator A fails the operator instance is recovered from the most recent checkpoint before the failure. Checkpoints are drawn as big dots in Figure 5.5 and the failure is marked with a bold cross. As Fig. 5.5 illustrates, operator A will start producing duplicate data stream elements after recovery until the point of the failure (in stream-time) is reached. After the failure ξ_f, the recovered operator instance is producing new data stream elements. The deterministic operator model presented in Chapter 4 guarantees that these duplicate data stream elements are identical with respect to stream time and payload information. Based on this model, the duplicates can therefore be dropped safely by the reliable stream transport facility of the DSM infrastructure. During the failure interval in global processing time a congestion of data stream elements has occurred at the sensor node which needs some time after recovery to be worked off again. This time interval in processing time needed to catch up with processing again is called catch-up time τ_c (see Fig 5.3). As this failure scenario shows the recovery of the transfer state is not

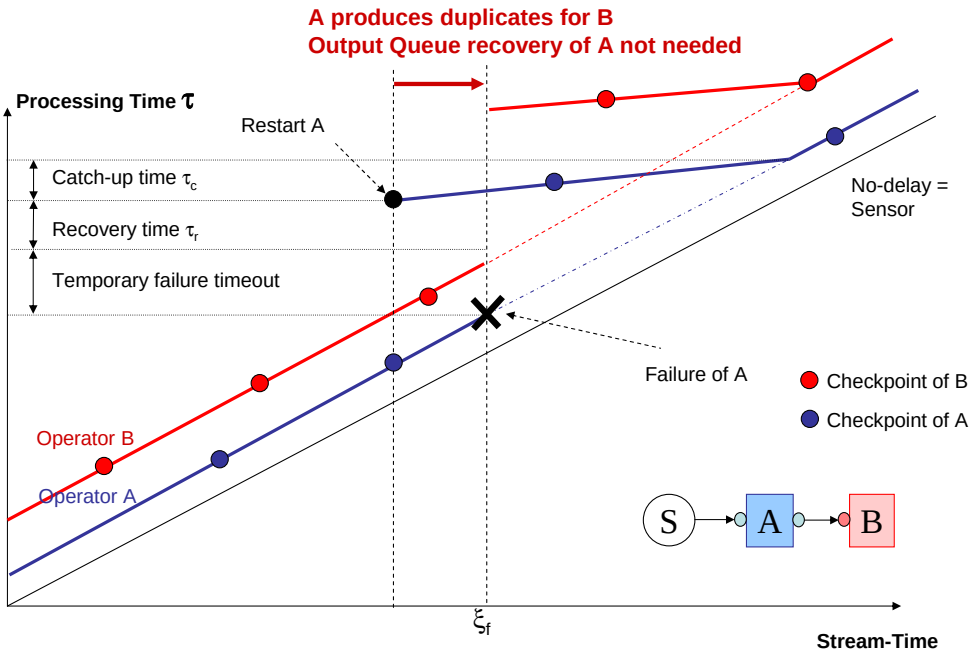


Figure 5.5: Single Failure Scenario

needed for single operator failures. In this case, recovered operator instances are even producing duplicates. Since the consuming operator instances do not fail in this scenario there is no need to retrieve data stream elements stored in the transfer state.

Unfortunately, for our intended applications multiple failure scenarios are very likely to happen and have to be considered in operator checkpointing. Figure 5.6 illustrates a *multiple operator failure scenario*. The stream process used for the example is the same as in the previous failure scenario. Now we assume that the two subsequent operators (A and B) fail at the same time. There can be several cases of ordering of two simultaneous failures. Simultaneous means, that both failure situations of the two operators overlap. Firstly, operator A failed before operator B but the recovery of operator A was not finished at the time when operator B failed too. After recovery of both operators from the most recent checkpoints, we see that operator A will recover in a state *after* operator B with respect to *stream time* of their connecting

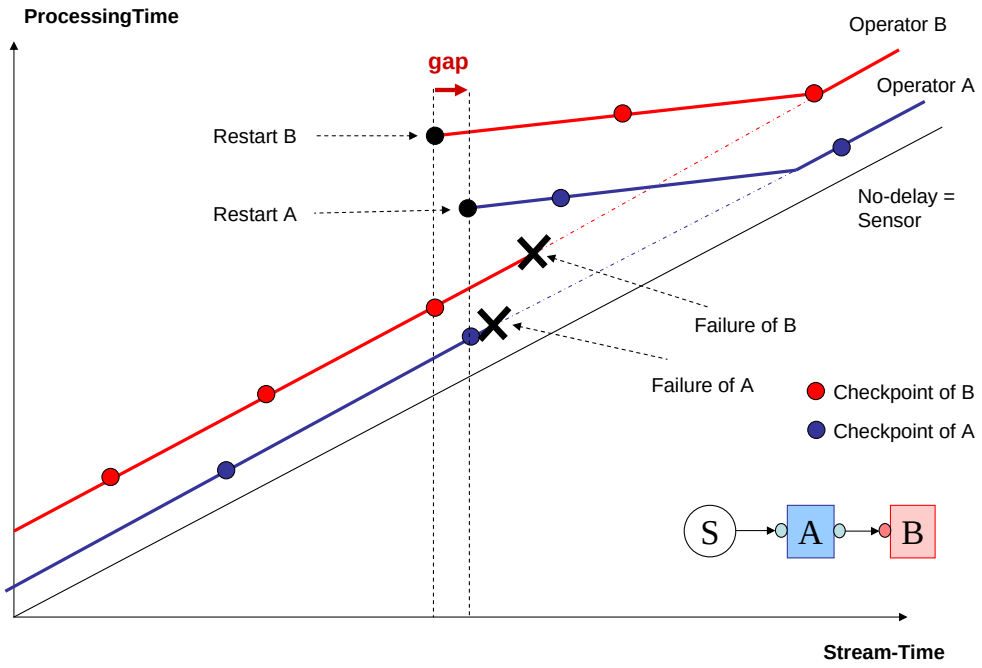


Figure 5.6: Multiple Failure Scenario

data stream. This means that operator B needs data stream elements to continue processing that have been produced by operator A *before* the most recent checkpoint used for recovery. In this case, we need the recovery of the transfer state of operator A because the transfer state at checkpoint time contains exactly those missing data stream elements. Otherwise, operator B is facing a gap after recovery. This gap will result in loss of data stream elements which is not allowed for the intended lossless reliability level. Moreover, Fig. 5.6 illustrates if moving the most recent checkpoints of operator A a little bit back so that the checkpoints of operator A and operator B are at the *same point in stream time* will omit the need for transfer state recovery.

Concluding the investigations so far, we see that recovery of the transfer state is not necessary for the single failure scenario. In the multiple failure scenario, we have to check whether the checkpoint of a subsequent operator (like operator B in Fig. 5.6) is before or after the checkpoint of the upstream operator A. If we can guarantee that the most recent checkpoint of a subse-

quent operator is later or equal than the most recent checkpoint of its predecessor at the backup node the checkpoint does not have to include the transfer state, which is highly beneficial in sense of *checkpointing overhead*.

5.8.2 Overhead of Operator Checkpoints

The overhead of operator checkpointing can be categorized in CPU, memory and network overhead. CPU overhead is caused by additional computational effort for scheduling and executing checkpoints. Memory overhead is mainly caused by data stream elements of the transfer state, which are kept in output queues until Check-Ack. Communication overhead is caused by the checkpoint messages sent from the operator provider to the backup provider. CPU and memory overhead is affordable. In general, modern mobile and embedded devices profit from higher memory density and more MIPS per Watt. Nevertheless communication overhead is still expensive because of its high energy consumption and therefore our main focus of optimization.

Taking a closer look into the communication overhead of standard checkpointing for DSM, we can estimate the *communication overhead* (CO) from *checkpoint message size* (CS) and *checkpoint interval* (CI):

$$\mathrm{CO} \propto \mathrm{CS}/\mathrm{CI}$$

This means that the communication overhead is proportional to the checkpoint message size CS multiplied by the checkpointing frequency $1/\mathrm{CI}$.

The checkpoint message, which contains the current operator state, consists of the operator state (s), the time context (TC), the transfer state (TS), and the routing information (RI). Therefore the checkpoint message size (CS) corresponds to:

$$\mathrm{CS} = \mathrm{sizeof}(\mathrm{s}) + \mathrm{sizeof}(\mathrm{TC}) + \mathrm{sizeof}(\mathrm{TS}) + \mathrm{sizeof}(\mathrm{RI})$$

Important to mention, the size of operator state, time context, and routing information are independent from the checkpoint interval. RI is even changing only infrequently and therefore is not included in each checkpoint message.

The size of the transfer state on the other hand is increasing with the checkpoint interval. But inversely shortening the checkpoint interval does not lead to unlimited smaller transfer state size in the checkpoint message. This is caused by the inherent delay between producing and consuming data stream elements. During the delay the producer may already produce additional elements before the consumer has received the previous ones. Therefore for checkpoint intervals that are in the range or smaller than this inherent processing delay the transfer state size is no more decreasing, but

sticking to this constant amount. Figure 5.7 illustrates this behavior. The point marked with MCI indicates the minimal checkpoint interval where the minimal size of transfer state (MSTS) is reached.

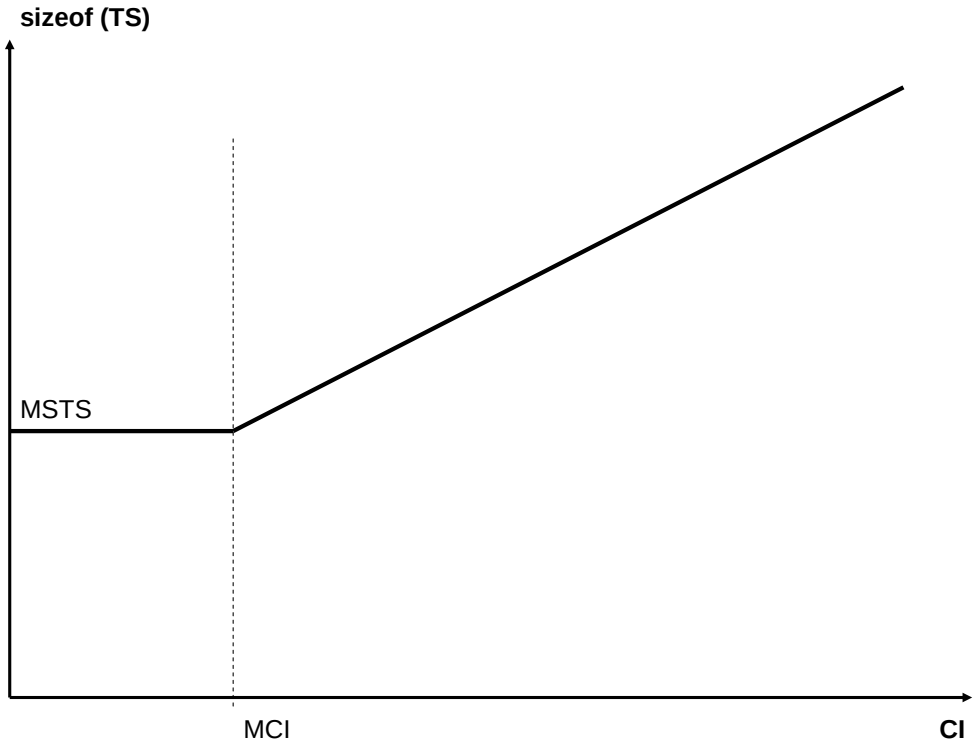


Figure 5.7: Transfer State Size

In the following, we call the function (illustrated in Fig. 5.7) between transfer state size and checkpointing interval $\mathrm{TSS}(\mathrm{CI}) = sizeof(\mathrm{TS})$. We can now derive the checkpointing overhead as:

$$\mathrm{CO} \propto (sizeof(\mathrm{s}) + sizeof(\mathrm{TC}))/\mathrm{CI} + \mathrm{TSS}(\mathrm{CI})/\mathrm{CI}$$

Figure 5.8 illustrates CO and the two terms of the previous formula as function of CI. The dashed line in this figure represents the overhead caused by the operator state and the time context ($(sizeof(\mathrm{s})+sizeof(\mathrm{TC}))/\mathrm{CI}$), whereas the dotted line represents the overhead caused by the transfer state $\mathrm{TSS}(\mathrm{CI})/\mathrm{CI}$. For reasonable checkpoint intervals ($\mathrm{CI} \geq$MCI), the majority of the checkpointing overhead is caused by the transfer state. Obviously, the difference in absolute values between the two is application dependent. But at

a qualitative level, we see that the above MCI the overhead caused by transfer state is constant whereas the overhead caused by operator state and time context is still decreasing with CI. Based on these consideration, we see it is very beneficial *to omit the transfer state* from the checkpoint messages if possible while keeping the requested reliability level.

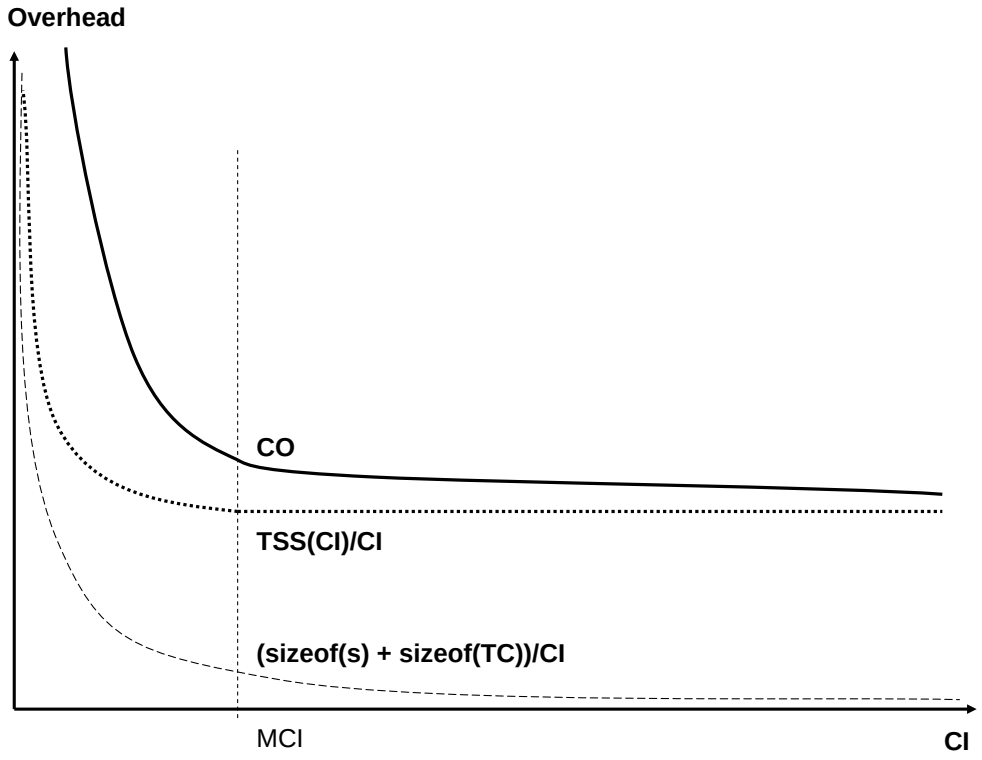


Figure 5.8: Checkpoint Overhead (CO)

5.9 Uncoordinated Operator Checkpointing

Uncoordinated operator checkpointing is a straightforward checkpointing algorithm which guarantees lossless reliability level for DSM. In uncoordinated checkpointing, a *local checkpoint scheduler* is generating independently *local checkpointing events* without coordination of checkpoints along interconnected operators.

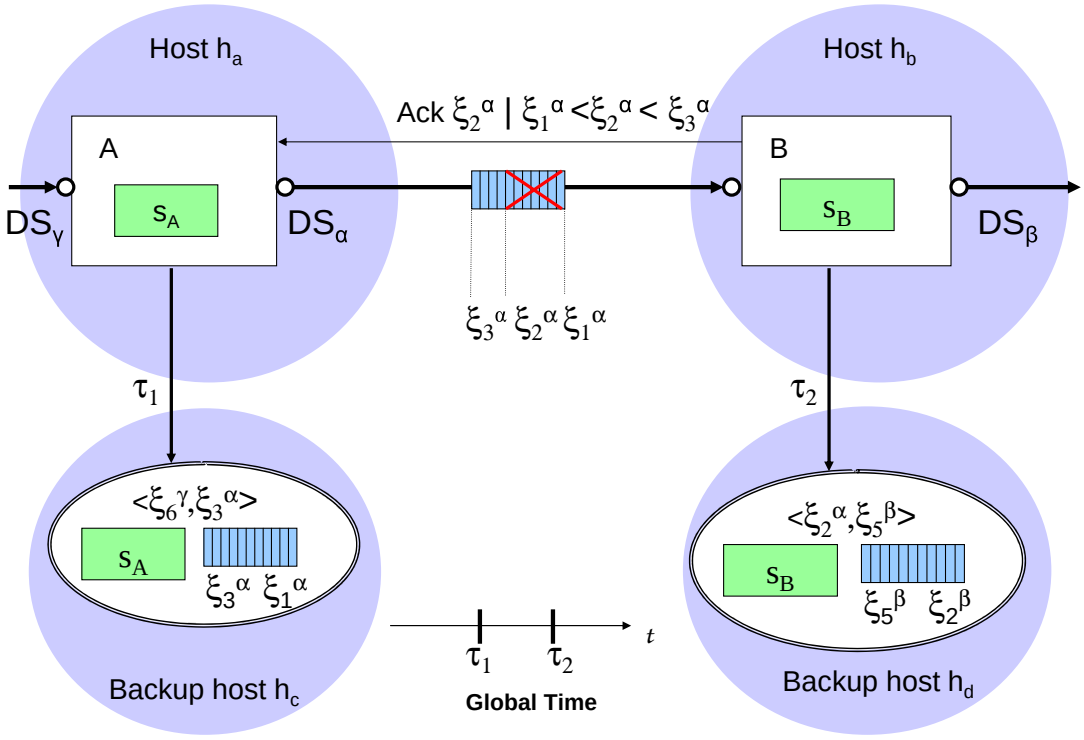


Figure 5.9: Uncoordinated Checkpointing

Figure 5.9 illustrates *uncoordinated checkpointing*. In the illustrated example, operator A schedules a checkpoint at τ_1. At this point in processing time, operator A has an operator state s_A, a time context $\langle \xi_6^\gamma, \xi_3^\alpha \rangle$ along the input data stream γ and the output data stream α, and a transfer state $TS_\alpha(\tau_1) = DS_\alpha(\xi_3^\alpha, \xi_1^\alpha)$. Later in time at τ_2 operator B schedules a checkpoint with operator state s_B, a time context $\left\langle \xi_2^\alpha, \xi_5^\beta \right\rangle$ along the input data stream α and the output data stream β, and a transfer state $TS_\beta(\tau_2) = DS_\beta(\xi_5^\beta, \xi_2^\beta)$. After performing this checkpoint, operator B will never need to rollback before this checkpoint so the transfer state of DS_α can be trimmed to the start stream time ξ_2^α which corresponds to the time context of the last checkpoint of B. This is done by sending an appropriate acknowledge message along all input edges after performing a checkpoint.

Figure 5.10 illustrates the pseudocode of the uncoordinated checkpointing algorithm. If the local checkpointing scheduler is triggering a checkpoint the checkpointing phase starts. During this phase, the current oper-

ator state, time context, and transfer states of each outgoing data stream are saved within a single checkpoint message. If the transmission of the checkpoint message to the backup host was successful, an appropriate Ack-message is sent along all incoming data streams including the current stream timestamp of the checkpoint along the edge. If Ack-messages are received, data stream elements before the included stream timestamp are marked for deletion. Moreover, if all downstream operator instances of a certain output port have sent acknowledgements, the data stream elements are deleted from the corresponding transfer state.

```
1   //This code is executed for each operator instance
2   //this is referencing to this operator instance
3   while (true)
4    if (local checkpointing event) then
5     //checkpointing phase
6     Store operator state s_this in checkpoint-message
7     foreach (output port op) do
8      Store transfer state TS of op in checkpoint-message
9     endforeach
10    Store time context TC of all input and output ports in checkpoint-message
11    Send checkpoint-message to backup-host (blocking with acknowledgement)
12    send Ack with ξ_ip^ack = ξ_ip^i | i ∈ (0, n] of checkpoint along input edges
13   endif
14   if (Ack ξ_op^ack received) then
15    Save Ack information for output port op
16    if (all downstream operators of output port op acknowledged)
17     Trim transfer state of output port op to ξ_s = ξ_op^ack
18    endif
19   endif
20  endwhile
```

Figure 5.10: Pseudocode of Uncoordinated Checkpointing

Lemma 5.20 Uncoordinated checkpointing as described above guarantees relaxed consistency in case of failures for which one or more operators are recovered from their recent checkpoints. Based on Lemma 5.19, this guarantees lossless and intra-stream order preserving reliability. ◇

Proof. In the *single failure case* only one operator fails. Without loss of generality, we choose operator B of Fig. 5.9 to fail. In general, operators have multiple data streams (edges) between them. The presented proof is shown for a particular edge but can be applied to each additional edge in the same way. In the single failure case, operator B is recovered from the last recent checkpoint taken at τ_2. Affected from the rollback to the previous checkpoints are the connected operators along DS_α and DS_β. For these two edges, we have

to proof that relaxed consistency is guaranteed. Firstly along the *input edge* DS_α, the transfer state has not been recovered and is starting from ξ_2^α. Since this was the stream-time of the checkpoint of B, the recovered operator B is able to seamlessly continue to work. Therefore, even the constraint for exact consistency is true:

$$(\xi_{op}^A = \xi_3^\alpha) \wedge (\xi_{ip}^B = \xi_2^\alpha)$$

Secondly along the *output edge* DS_β, the transfer state has been recovered and is starting from ξ_2^β to ξ_5^β. Since operator C, which is receiving DS_β, has not failed the following condition for relaxed consistency is true:

$$(\xi_{op}^B = \xi_5^\beta) \wedge (\xi_{ip}^C \geq \xi_2^\beta)$$

More input or output edges of an operator are proven in the same way. Resulting of this, we have proven the lemma for the single failure case.

For the *multiple failure case*, we start with a failure of two connected operators. Since consistency is defined pairwise, this argumentation can be extended to general multiple failure cases and multiple interconnecting edges. Going back to the example, we assume operator A and operator B have been recovered from their recent checkpoints. For the consistency evaluation of this case, we can distinguish between edges to non-failed operators and *edges between failed operators*. For edges to non-failed operators, we use the argumentation of the single failure case in order to proof relaxed consistency. In this case, we have only to investigate for relaxed consistency along the edge (data stream DS_α) connecting the failed operators A and B. The current stream time context along this edge is for the recovered operator A ξ_{op}^A and for the recovered operator B ξ_{ip}^B. Since operator A is responsible for the transfer state of DS_α, we have ξ_{op}^A guaranteed to be equal to the end of the transfer state ξ_e. Furthermore, $\xi_{ip}^B \geq \xi_s$ has to be valid. Which can be proven by distinguishing two cases. Firstly $\xi_{op}^A \leq \xi_{ip}^B$, in this case $\xi_{ip}^B \geq \xi_s$ because $\xi_s \leq \xi_e$ and $\xi_{op}^A = \xi_e$. Secondly $\xi_{op}^A > \xi_{ip}^B$, in this case the described acknowledge mechanism guarantees $\xi_{ip}^B \geq \xi_s$, because only after a checkpoint ξ_s is trimmed to later stream time. Therefore, the constraint for relaxed consistency is valid:

$$(\xi_{op}^A = \xi_e) \wedge (\xi_{ip}^B \geq \xi_s)$$

Other edges between failed operators are evaluated in the same way. □

The presented uncoordinated checkpointing algorithm is suffering from a high transport overhead on sending checkpoint messages from the active host to the backup host. Transfer overhead is defined as the relation between the transport load because of checkpoint messages compared to the transport load of a DSMS without checkpointing. Since the transfer state is part of the

checkpoint message and the transfer state along an edge can only be trimmed when the receiving operator itself has performed a checkpoint, this algorithm leads to a transport overhead of more than 100%. This is due to every data stream element that is sent is guaranteed to be in a transfer state for at least one checkpoint. In the investigation of this case, we discover a high overhead of checkpointing mainly resulting from checkpointing the transfer state. The high cost for checkpointing of the transfer state compared to the cost of checkpointing of the operator state comes from a basic assumption generally applicable to DSM processing: Each operator that continuously processes on incoming data streams is extracting relevant information. Therefore the information content or entropy of the input data stream is naturally lower than of output data streams with respect to relevant information. Based on this fact, we assume a current operator state can be encoded by using less bytes compared to the overall input data streams that have produced the current operator state.

In order to supersede the transfer state checkpointing problem, we present in the following an efficient coordinated checkpointing (ECOC) algortihm, which is omitting transfer states from checkpoint messages by still keeping the requested reliability levels based on relaxed consistency.

5.10 Efficient Coordinated Operator Checkpointing

The design goal of an efficient checkpointing algorithm is to reduce the overhead needed for checkpointing messages between the active and the backup host. A great portion of the checkpoint message are transfer states, which are needed to guarantee relaxed consistency. In order to develop an algorithm, which is allowed to omit transfer states from checkpointing, the following lemma describes when the recovery of transfer state is not needed:

Lemma 5.21 The recovery of transfer state is not needed to achieve relaxed consistency in case of failures if it is guaranteed, that $\xi^{x}_{op} \leq \xi^{y}_{ip}$ for every pair of checkpoints along an edge connecting two operators (x, y) in a stream process. ◇

Proof. As in the previous proof of Lemma 5.20, we distinguish between three different kinds of edges along which transfer state may be recovered. As for the previous proof, the argumentation can be applied to several input, output, and connecting edges.

Firstly for the *input edge* of an affected, failed operator, the transfer state has no to be recovered because for this kind of edge the producing operator who is in charge of keeping the transfer state has not failed.

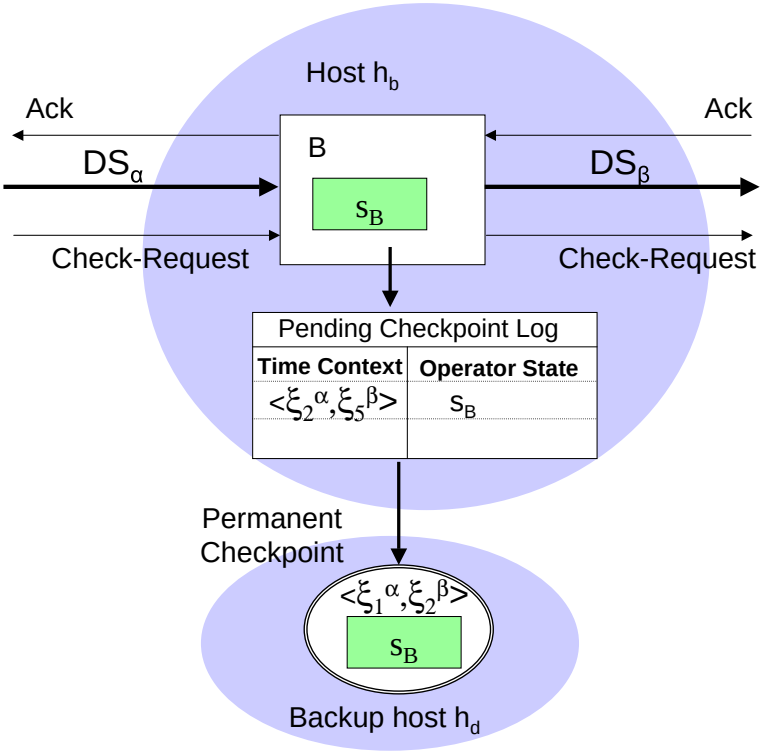


Figure 5.11: ECOC Overview

Secondly for the *output edge* of an affected, failed operator, the transfer state is to be recovered. But this recovery is not necessary because the destination operator has not failed and therefore even expects newer data stream elements, than the one produced by the recovered, failed operator. Relaxed consistency allows for ignoring these incoming duplicates.

Thirdly for the *connecting edge* of two affected, failed operators (x, y), depending on the time context of the checkpoint of the source operator ξ_{op}^{x} along the connecting edge to the checkpoint of the destination operator ξ_{ip}^{y}, we can define two different cases: Firstly $\xi_{op}^{x} \leq \xi_{ip}^{y}$, in this case recovery of the transfer state is not needed because operator y is expecting even newer elements as operator x is producing. Secondly $\xi_{op}^{x} > \xi_{ip}^{y}$, in this case recovery of the transfer state is needed because operator y is expecting elements older than the elements as operator x is producing after recovery. □

Efficient coordinated operator checkpointing (ECOC) is our proposed checkpoint algorithm to guarantee this lemma. In this case, transfer state

content can be safely omitted from checkpoint messages because the transfer state is not needed for recovery even in case of multiple failures. In order to achieve a temporal coordination of checkpoints, ECOC introduces an additional *checkpoint-request message (Check-Request)* which is sent to downstream operators attached on the corresponding data stream element extending the alphabet of the payload information. Figure 5.11 illustrates the messages used for checkpoint coordination and the additional *pending checkpoint log* (PCL). An additional advantage of this approach is that message exchange is only between connected operator instances in a Peer-to-Peer fashion without centralized control. The pending checkpoint log is used to store checkpoints locally until along all outgoing edges all downstream operator have performed their checkpoints and $\xi_{op}^{x} \leq \xi_{ip}^{y}$ is fulfilled. For this reason, a two-phase protocol described in pseudocode in Figure 5.12 is applied. Checkpoints may be triggered by a *local scheduler* or by a Check-Request message from an upstream operator. The local scheduler can follow different strategies for checkpoint planning, e.g., every 50 incoming data stream elements.

In the first phase (*planning phase*), a checkpoint is triggered either by receiving a Check-Request or by the local scheduler and stored in the local PCL. Additionally, along all output edges Check-Request messages are sent with the corresponding stream-time context ξ_{op}^{i}. In the second phase (*checkpoint phase*), corresponding Ack messages with ξ_{op}^{ack} are received along the outgoing edges. If for a checkpoint in PCL all output edges have received the Ack messages where $\xi_{op}^{i} \leq \xi_{op}^{ack}$ the checkpoint is *fully acknowledged*. In this case, the checkpoint is sent to the backup host and removed from the PCL. The PCL is a data structure in the local memory assigned to an operator instance holding a list of checkpoints ordered by time of creation. Following transitively the edges, we see that checkpoints requests are cascaded until the reach an operator instance without internal output data streams because of the outside receiver assumption (see Section 4.2). Outside output data streams are ignored in this case. At this operator instance the checkpoint can be passed immediately to the backup host. The Ack messages are cascaded back transitively against the flow of data streams and allow to make pending checkpoints permanent at backup hosts.

A drawback of the ECOC approach is the delay of checkpoints in the planning phase. Checkpoints are delayed until all downstream operators have acknowledged the checkpoint. After acknowledgement the checkpoint message is sent to the backup node. In particular, these delays are getting longer if we go upstream, closer to the sensors, in a stream process. Downstream, closer to the final consuming operators, which itself have no more output streams, the delays are getting shorter. These delays are not blocking stream processing and have no affect on time constraints in stream time. Assuming the case of

```
1   //This code is executed for each operator instance
2   //this is referencing to this operator instance
3   while (true)
4    if (Check-Request) or (local checkpointing event) then
5     //planning phase
6     if |{e|x = this ∧ y ≠ ow}| > 0 then
7      add new pending checkpoint do PCL
8      send Check-Requests to all output streams on all m output ports with
     ξ^i_op | i ∈ (0, m] ∧ y ≠ ow
9     else
10     do permanent checkpoint
11     send Ack's upstream ξ^ack_ip = ξ^i_ip | i ∈ (0, n] of checkpoint along all input edges
12    endif
13   endif
14   if (Ack ξ^ack_op received) then
15    //checkpoint phase
16    Save Ack information for output port op
17    if (all downstream operators of output port op acknowledged)
18     Trim transfer state of output port op to ξ_s = ξ^ack_op
19     foreach (checkpoint in PCL) do
20      if checkpoint is fully acknowledged then
21       save checkpoint permanently at backup host
22       remove checkpoint from PCL;
23       send Ack with ξ^ack_ip = ξ^i_ip | i ∈ (0, n] of checkpoint along input edges
24      endif
25     endforeach
26    endif
27   endif
28  endwhile
```

Figure 5.12: Pseudocode of ECOC

a failure in the planning phase, the affected operator is recovered from the most recent permanent checkpoint before. In this case, correct data stream processing is still guaranteed, but more duplicates are produced because of recovering from the older checkpoint resulting in longer time for catchup. On the other hand, storing checkpoints in the pending checkpoint log, requires additional memory overhead. Since we do not need to store the transfer state in the pending checkpoint log, this overhead is similar to the reduced communication overhead. Therefore, we consider these drawbacks as acceptable.

Lemma 5.22 The ECOC algorithm presented in Section 5.10 guarantees relaxed consistency for non-cyclic data stream process graphs if in case of failures one or more operators are recovered from their recent checkpoints. Based on Lemma 5.19 this guarantees lossless and intra-stream order reliability. ◇

Proof. ECOC ensures coordination of checkpoints along outgoing edges which guarantees Lemma 5.21. Combined with the proof of Lemma 5.20 for uncoordinated checkpointing, this ensures relaxed consistency for the stream process when recovering from one or more failures. Furthermore, it is needed to proof that ECOC terminates when cascading of checkpoint requests transitively along connected edges. This is guaranteed because finally each path in a non-cyclic stream process graph will reach an outside world vertex. The last operator of the DSMS is allowed to perform immediately checkpoints at the backup host. This is valid because of the assumptions on the outside world defined in Section 4.2. Firstly, the outside world never fails and secondly the *outside receiver assumption* allows to re-sent data stream elements, which is caused when an operator instance sending to the outside world is recovering from a checkpoint. Finally, since the Ack-messages caused by permanent checkpoints are cascaded upstream in the same manner, all pending checkpoints will be acknowledged, which proofs the termination of ECOC for non-cyclic stream process graphs. □

So far, the presented ECOC algorithm does not support *cycles* in the graph of stream processes, since Checkpoint-Request messages will continuously trigger new pending checkpoints in the cycle and pending checkpoints will never be acknowledged. Optimizations of ECOC for complex stream process graphs including arbitrary combinations of splits, joins, and cycles are addressed in the following section.

5.11 Extensions of ECOC for Joins and Cycles

Supporting complex stream processing topologies is crucial for real-world DSM applications. Recently, research in the area of DSM is focusing on adaptive stream processing [LZJ+05, CKP+05, BMM+04, YTP05]. In this research projects, stream processing is continuously adapting to different environment conditions e.g, system load, sensor input characteristics. In general, this implies that a feedback loop is applied within a stream process graph where results of current stream processing are affecting the stream processing processing in the future. In order to model and support such feedback cycles for DSM processing also reliability algorithms have to support these topologies.

Firstly, we focus on optimizations for join-operators. Particularly for join-operators, obeying all Check-Request messages that may come along different input edges will increase the checkpoint frequency at the operator itself and subsequently on all operators following transitively downstream in the data stream process graph. The checkpoint requests along the different in-

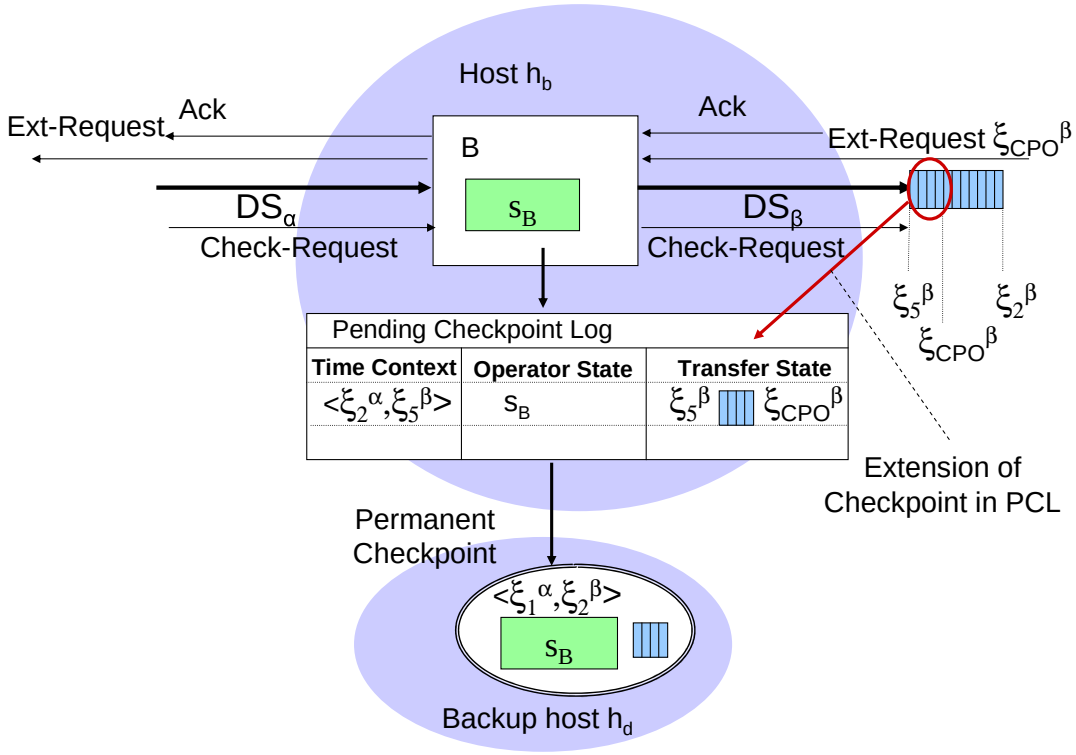


Figure 5.13: Extended ECOC

put edges are not correlated and therefore may be received shortly after each other. An increased checkpoint frequency may reduce the benefit achieved by ECOC because resulting again in increased checkpoint overhead.

In cases where multiple checkpoints are requested in a *short time interval* it may be beneficial *to extend* the previous still pending checkpoint *by the necessary transfer state* instead of performing a new checkpoint. Multiple checkpoints in a short time frame may appear due to multiple input edges on operators (as for join operators) or when local scheduling of checkpoints is combined with obeying Check-Request messages from input edges. Due to the coordination of checkpoints in ECOC, a checkpoint imposes load both on the operator itself and all operators transitively following operators along downstream paths in the data flow of the stream process.

In order to reduce the overhead for checkpoints triggered within a short time frame, we propose an optimized version of ECOC (see Fig. 5.13), where an additional *extension request message (Ext-Request)* is introduced to request the extension of an existing checkpoint in the PCL by a limited part of the

```
1   |TS_e| = sizeof {ds} * (ξ_CPN − ξ_CPO)
2   checkpointLoad = ACPS
3   foreach output edge i do
4    checkpointLoad = checkpointLoad + ADCPS_i
5   endforeach
6    attach checkpointLoad to own ACK messages
7    //check if extension is beneficial
8   if (extTransferSize < checkpointLoad) then
9    Ext-Request from ξ_CPN to ξ_CPO
10  else
11   perform a new checkpoint at ξ_CPN
12  endif
```

Figure 5.14: Pseudocode Optimized ECOC

transfer state. Adding a subset from ξ_{CPO} to ξ_{CPN} of the transfer state of checkpoint CPN extends the relaxed reliability constraint from $\xi_{CPN} \leq \xi^{i}_{ip}$ to $\xi_{CPO} \leq \xi^{i}_{ip}$, where ξ_{CPO} is before ξ_{CPN} which is the time context of the checkpoint to be extended. In Figure 5.13 ξ^{β}_{5} is the stream time of the pending checkpoint with respect to output stream β and ξ^{β}_{CPO} is the stream timestamp of the Ext-Request. This extension is only applied if the *overall checkpoint load* of the system is reduced compared to the standard ECOC algorithm. Based on this, the extension is only done if the size of the extended transfer state $|TS_e|$ is smaller than the overhead caused by performing a new coordinated checkpoint.

As Fig. 5.14 illustrates, *optimized ECOC* allows the receiver of a Check-Request message to decide whether a new checkpoint is performed or an Ext-Request is returned to the sender. The $ACPS$ is the average local load imposed by a checkpoint acquired during runtime of an operator instance. The $ADCPS_i$ is the average downstream load imposed by all transitive checkpoints triggered along the corresponding output edge i. This statistic is passed upstream as attachment on Ack-messages.

Still, ECOC has to support *cycles* in the stream process. A closed control cycle in a stream process is in particular beneficial in scenarios where stream processing has to adapt dynamically to changes in the data stream characteristics during runtime [LZJ+05, CKP+05, BMM+04, YTP05]. For example, the processing of an ECG signal have to be changed when the heart beat becomes pathological. In order to adapt stream processing, the operator parameters have to be changed. The cycle support is based on the previous optimized ECOC algorithm. Fig. 5.15 illustrates a cycle in a stream process. The cycle caused an infinite cascading of checkpoint requests with unlimited increase of the pending checkpoint log without applying permanent checkpoints at the

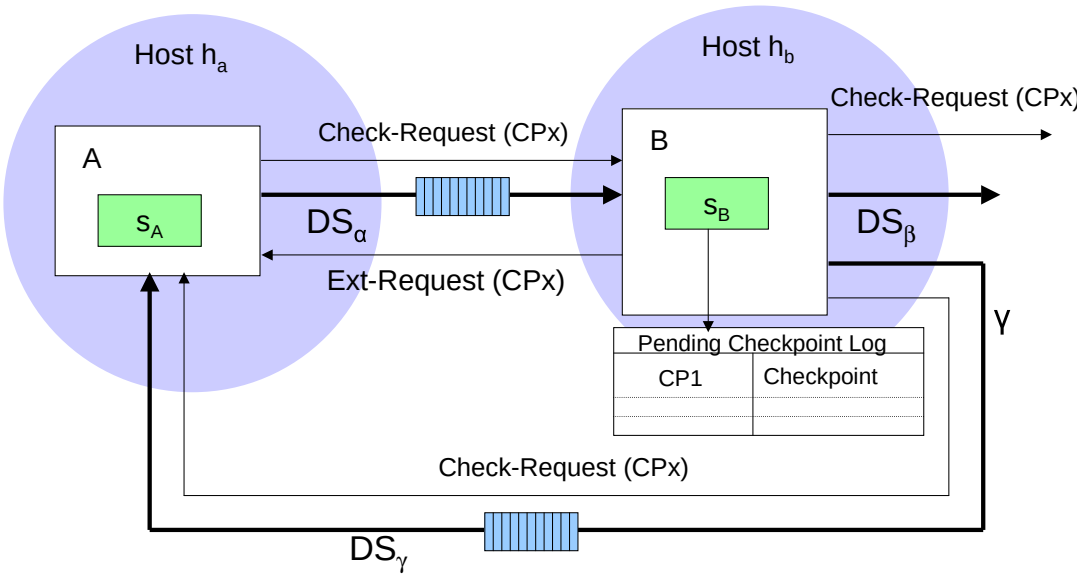


Figure 5.15: Cycles with Optimized ECOC

backup host. To break this infinite cycle, unique identifiers are applied to Check-Request messages by the first operator instance that triggers the coordinated checkpoint. All transitive checkpoints caused by cascading checkpoints inherit the same identifiers. Consequently, the checkpoint identifiers are also used to identify checkpoints in the pending checkpoint log. Therefore, whenever a Check-Request message is received which has a checkpoint identifier that already is available in the pending checkpoint log a cycle in checkpoint coordination is detected. After the cycle is detected the affected node can easily break the request cycle by requesting an extension of the previous checkpoint by using the presented extended ECOC approach. Obviously, the pending checkpoint is extended in this case without regard to checkpoint load statistics.

Lemma 5.23 The extended ECOC algorithm presented in Section 5.11 guarantees relaxed consistency for cyclic and non-cyclic data stream process graphs if in case of failures one or more operators are recovered from their

recent checkpoints which may also be extended checkpoints. Based on Lemma 5.19 this guarantees lossless and intra-stream order reliability. ◇

Proof. Extended ECOC ensures coordination of checkpoints along outgoing edges. Based on the proof of Lemma 5.22, this ensures relaxed consistency for the stream process when recovering from one or more failures in a non-cyclic data stream process without extended checkpoints.

In addition, it is needed to proof that extended ECOC is also correct for extended checkpoints. The extension of a checkpoint means that a checkpoint is enriched with the transfer state as in the uncoordinated case. For this reason, we can refer to the proof of Lemma 5.20 for uncoordinated checkpointing on any edge along a Ext-Request has been performed. Exemplary, we investigate the edge between operator B and operator C, which follows along DS_β. The following condition for relaxed consistency is true:

$$(\xi_{op}^{B} = \xi_{5}^{\beta}) \wedge (\xi_{ip}^{C} \geq \xi_{CPO}^{\beta})$$

As in the proof of Lemma 5.20, we can investigate that relaxed consistency is guaranteed for single and multiple failure cases along any edge within a data stream graph.

Furthermore, it is needed to proof that extended ECOC also terminates when cascading of checkpoint requests are passed transitively along connected edges for cyclic stream process graphs. This is guaranteed because any checkpoint request in a cyclic stream process branch will finally reach the origin of the request again. In this case, the cycle is detected and an extension of the checkpoint is performed. In all other cases, non-cyclic branches within the stream process graph will cause termination of the checkpoint request when reaching outside world vertices as shown in the proof of Lemma 5.22. □

Finally, the proposed optimized ECOC approach is able to support lossless reliability of DSM at operator level with affordable effort. Efficient reliability is achieved for complex stream process graphs including a large number of operators with combinations of splits, joins, and even loops. Optimized ECOC is able to adapt its behavior according to acquired data stream statistics in a way that the overall checkpointing overhead is kept minimal based on Peer-to-Peer communication within neighboring operator instances without establishing a centralized checkpoint control.

6

Implementation

6.1 The OSIRIS Infrastructure Implementation

OSIRIS (Open Service Infrastructure for Reliable and Integrated process Support) [SWSS04, SST+05, SWSS03] is a prototype implementation of a Hyperdatabase infrastructure (see Section 3.1). OSIRIS has been developed at the *Database Research Group* at ETH Zurich and has been constantly evolved at University for Health Sciences, Medical Informatics and Technology (UMIT) and University of Basel. OSIRIS is a platform that allows combining different distributed services into processes. Each participating host in the OSIRIS network has a prototype implementation of the HDB-layer middleware, called OSIRIS-layer, running locally.

Moreover, OSIRIS is equipped with the O'GRAPE (OSIRIS GRAphical Process Editor) [WSN+03], a Java-based process modeling tool that supports a designer in defining and validating processes. It allows for easy creation of process descriptions without programming skills. In addition, O'GRAPE supports the integration of existing application services by leveraging existing Web service standards like SOAP and WSDL.

6.1.1 Implementation Details of the OSIRIS Infrastructure

The original OSIRIS infrastructure has been implemented at ETH Zurich in C++ and including *ISIS* [MST04] consisting of more than 300.000 lines of code. ISIS stands for *Interactice SImilarity Search* and is a prototype application for information retrieval in multimedia collections based on the OSIRIS infrastructure. This implementation relies on using Microsoft Visual C++ Studio [Micb] as programming environment and due to this fact the execution

platform is limited to Microsoft Windows systems. Unfortunately, also Windows Mobile is not supported in unmanaged C++. The fact that the intended application scenario depends on the use of embedded and mobile devices has demanded to re-implement the basic OSIRIS process management functionality in Java for this thesis. The Java version of OSIRIS has been developed at UMIT and University of Basel and is based on Java 2 SE 1.3, which is available for a variety of different platforms like Windows, Linux, and also on mobile devices running Windows Mobile. Therefore, the only prerequisite of the Java version is to have an J2SE 1.3 compliant Java virtual machine available at the desired target system. The Java version contains about 20.000 lines of code, without counting third-party, open-source libraries, like Axis, DOM4J, or Jetty. The Java version can make use of a variety of JDBC-enabled databases, like Microsoft SQL Server [Mica], MySQL [MyS], or Derby [Der]. In particular Derby is beneficial to use because it is fully programmed in Java and can run in embedded mode within the same virtual machine as OSIRIS, so there is no need to install and configure a separate database server on the node. Moreover, Derby has a very small footprint (about 2 MB) and still offering transactions and sufficiently supporting SQL92. For this reason, Derby is even able to run on a Windows Mobile PDA with acceptable performance. Due to the remarkable support for Java in the field of Web services and application servers, we have integrated a Web service framework within the Web component of the Java version. Technically, this is done by integration of the embedded open-source Java application server Jetty [Jet] and Axis [Axi] running as part of a system service. This framework allows to host Java Web services within our OSIRIS infrastructure and control their life-cycle and load situation, which offers a higher degree of control for loosely coupled services. Also in this case, we support mobile devices, like Windows Mobile machines, due to the small footprint required by Jetty and Axis.

In the following, we describe the most important functionality of OSIRIS with respect to decentralized peer-to-peer process execution.

- **Messaging:** The local OSIRIS-layer allows for a reliable exchange of messages between all nodes of the OSIRIS network. In addition to TCP transport, the peer-to-peer messaging framework JXTA [JXT] has been incorporated as additional transport layer to abstract from networking obstacles, like inconsistent addressing and communication problems due to firewalls or heterogenous network environments.

- **Horus:** The *Horus* component is the agent of a service provider and is responsible for all external communication. The Horus is responsible for activation and deactivation of local services. External communication is done via two pipelines for incoming and outgoing messages, respectively.

Pluggable handlers are applied to the pipeline and offer specific processing support for messages. In particular, process execution is done by exchanging process messages, which are processed by a dedicated process handler.

- **Process Handler:** The *process handler* is plugged in the incoming message pipeline of the Horus (see Fig. 6.1) and executes local process activities (i.e., locally invokes a service) based on information in process messages and replicated execution units of the current process. After local activity execution, the process message is updated (e.g., with results of a service invocation) and forwarded to all subsequent activities (providers).

- **Replication Manager:** Meta data replication is based on a hierarchical organization with global repositories and clients replicating from them. The replication manager, which is part of the OSIRIS-layer, is keeping local replicas consistent according to configured subscription settings. Replication management is based on publish/subscribe techniques. The primary copy resides at the global repository. Each local OSIRIS-layer as client that needs replicas has to subscribe for the selected information. As a result of this subscription, the repository publishes the current information. Whenever the primary copy changes, updates are published to all subscribed clients. The following three constraints on meta information explain, how replication can be made affordable. First, updates on certain information (e.g., process definitions) are infrequent. Second, nodes only require parts of the global information (e.g., only about pieces of processes that the provider is able to host). Lastly, changes on global meta information are not always critical (e.g., small changes on load information). This is specified by a *freshness level* attached to the subscription settings. According to this, updates of marginal changes may be skipped as long as the replica is sufficiently close to the primary copy. Additionally to classical publish-subscribe, in OSIRIS a subscriber is also able to insert or update new information in the local repository and these updates are propagated back to the global repository. Update messages only contain changed parts of the subscribed information to reduce replication overhead.

- **Routing Handler:** The *routing handler* is a handler in the outgoing message pipeline of the Horus and responsible for selection of an appropriate provider as destination of outgoing messages. If multiple providers are able to process the corresponding message, the best suited

according to current load (the locally replicated approximation) is selected (load balancing).

- **Process Component:** The *process component* is a local OSIRIS system service managed by the Horus, which is in charge of starting the execution of a process within the OSIRIS network. The process component first looks up the necessary activities within the process control flow and creates the required process messages which are delivered to appropriate providers. The process component is also responsible for feeding the process messages with the correct input parameters and for collecting the process result parameter(s). The process result is returned to the calling application. From the calling application's point of view, a process execution is just a service invocation. This mechanism allows also for the easy nesting of processes as services within other processes.

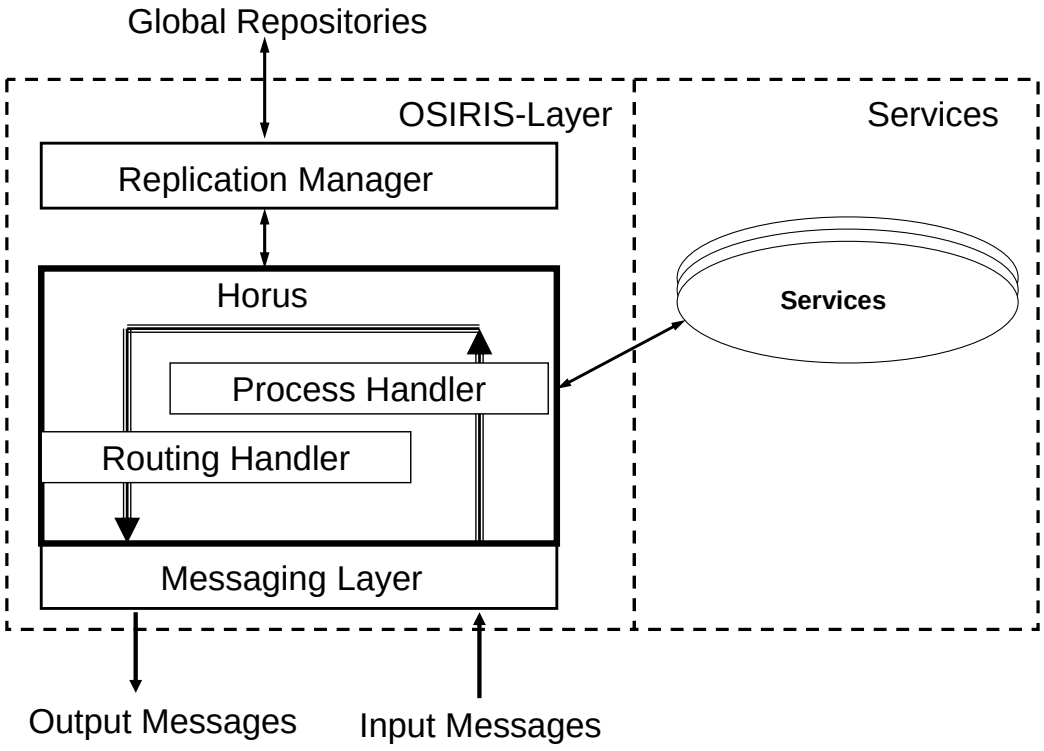


Figure 6.1: The OSIRIS Layer

6.1.2 Process Execution within OSIRIS

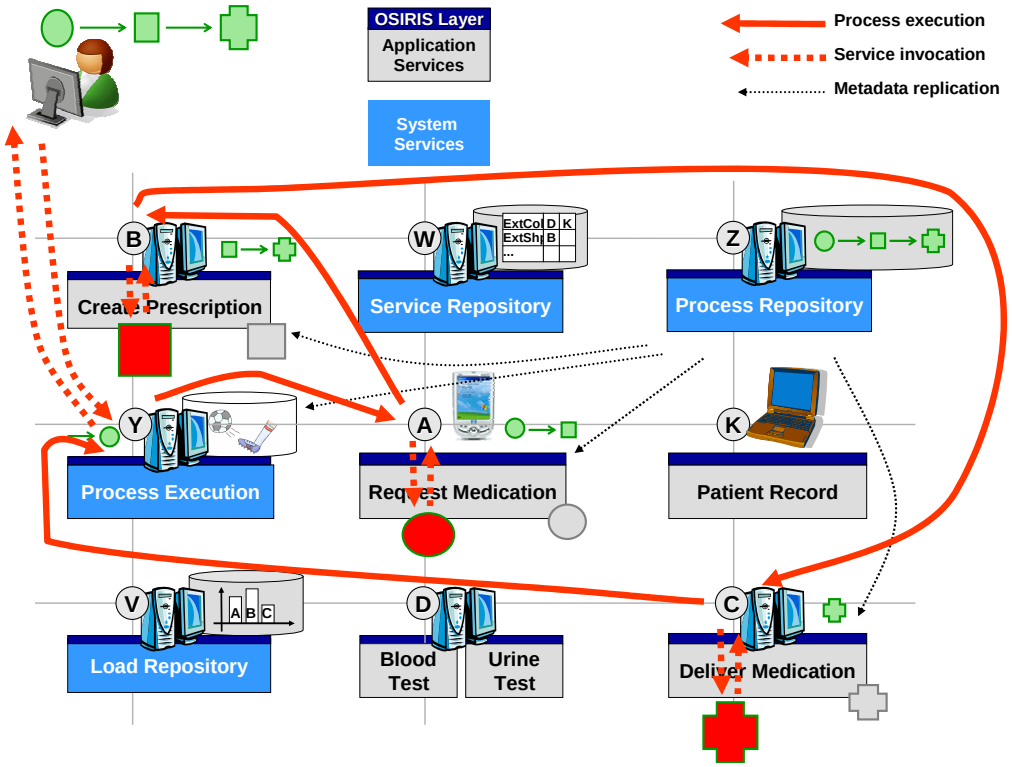


Figure 6.2: The OSIRIS Process Execution

Figure 6.2 illustrates the distributed process execution in the OSIRIS prototype with a process containing three subsequent activities in the control flow. The executed process is the example process of Fig. 3.1 again and the application services are illustrated by the different green shapes. System services are depicted in the middle (global repositories) and, in addition, each node in the network is equipped with a local OSIRIS-layer. It can be seen that process execution (solid, bold arcs) is decentralized and that all services are invoked locally (dashed, bold arcs). Meta data replication from global repositories to local OSIRIS-layers is decoupled from process execution (dashed, thin arcs). The invocation of the example process is done by the process component which is generating an appropriate process messages, which is a serialized form of a process instance. The process message is send to an provider that is able to offer the first service within the control flow. By passing the process message the process component is no longer in charge of the process

instance. By receiving the process message the provider gets in charge of the process instance and executes the local service as an activity of the process. Input parameters for the service are derived from the *whiteboard* of the process instance. The whiteboard hosts process variables and is used to model the data flow between the services. Accordingly, result parameters of the service invocation are mapped to the whiteboard again. The whiteboard itself is part of the process instance and serialized within the process message. The mappings between whiteboard variables and service parameter are part of the process definition.

6.2 The OSIRIS-SE Infrastructure Implementation

In this section, we describe our implementation of the proposed reliable DSM infrastructure (see Section 3.2), which has been implemented on top of the re-implemented Java version of the existing infrastructure prototype OSIRIS. OSIRIS-SE (Stream Enabled) [BS07, BSS05, BSS04] supports integrated stream processing and process management in order to achieve flexibility and reliability for monitoring applications in healthcare. Stream processing within OSIRIS-SE is done by execution of *stream processes*.

6.2.1 Global Repositories

In the following, we explain how the global repositories that have been implemented in order to support DSM in a peer-to-peer fashion.

- **Process Repository** holds the global definitions of all processes, both traditional and streaming. Stream process definitions are decomposed into *transmission units (TUs)*. For details on TUs see Section 3.2.

- **Subscription Repository** records all online providers within the OSIRIS-SE network. Moreover, the repository keeps a list of all available services and operator types, offered by these providers. Again, as for conventional process management also in DSM, providers hosting operators only need to know subscription information about providers offering subsequent operators with respect to the available stream process definitions. Firstly, all TU's are replicated to providers which offer a producing operator type. Secondly, the subscription information is replicated based based on the consuming operator types within the replicated TU's.

- **Operator Backup Repository** is storing information about providers in the network that are responsible to keep a backup of a currently running operator instance. Operator checkpoints are needed for reliable stream process execution and are explained in detail in Chapter 4. This information is published to providers that are potential backup provider in case the operator instances fails at its current provider.
- **Join Repository** holds the current providers of all running operators with more than one input stream, also known as joins. Since preceding operators may be running on different providers, a designated provider is needed to make the routing decision for the join such that the different streams actually arrive at the same provider for being joined. The designated provider is selected by the routing flag of the corresponding TU. After routing, the decision has to be published to the other providers via the join repository. A repository entry contains process ID, operator ID, and actual provider address.

6.2.2 Extended OSIRIS-Layer

Each service provider has an OSIRIS-layer running locally for meta data replication and management, operator invocation, etc. For reliably dealing with data streams, this layer has to be extended by the following additional modules. Figure 6.3 illustrates the extended OSIRIS-layer modules necessary for data stream management.

- **Stream Transport (ST)** implements a reliable data stream communication between two OSIRIS-SE providers based on the functionality provided by the message layer of OSIRIS. ST realizes the reliable transport of data streams with proper order of elements (*FIFO*) and detection of gaps. This module can leverage network dependent features to improve transmission of data streams (e.g., produce optimal message sizes by combining or splitting messages). Moreover ST handles the in- and output queues for data stream elements to keep the transfer state, enforces the correct order of data stream elements based on their sequence numbers, requests missing data elements from senders, and eliminates duplicates. In case of loss on transmission or failure of output providers, elements are re-sent. These issues are realized by acknowledgement protocols between sender and receiver. Additionally, ST checks as by-product whether all receivers are alive.
- **Operator Management (OM)** connects the incoming and outgoing data streams with the corresponding local operators, establishes internal connections between local operators. Internal connections are

needed if more operators of the same stream process are running on the same provider. Internal connections are established directly without need of ST. Usually for internal connections, the request for transmission of data stream elements originates from the sender in *Push*-style. *Pull*-style communication where the receiver triggers the transmission is implemented for sensor-operators. Sensor-operators are querying a sensor therefore the information flow is coming from the outside world. Necessary routing information is provided by the Stream Process Manager. OM also takes checkpoints of local operator instances, which are necessary for migrating operator execution to other service providers in case of high load or failure.

- **Stream Process Manager (SPM)** is responsible for activation and deactivation of operator instances, migration of operators in case of high load or failure, routing of data streams by selecting the provider of a subsequent operator, doing backups of operator state checkpoints at the backup host, and invocation of failure handling processes. For accessing the state of a running operator instance two *system data streams* are implemented. Firstly, the *config stream* allows to configure a previously instantiated operators. This is used for the initial configuration of operator instances with parameters defined in the corresponding stream process definition. Furthermore, the config stream can be used to alter dynamically state parameters of an operator during runtime. Secondly, the *state backup stream* allows to retrieve the current operator state. This is particularly necessary for checkpointing. The config stream is implemented in Push-style and the state backup stream is implemented in Pull-style. Therefore, the SPM is triggering both configuration and checkpointing of running operator instances.

6.2.3 OSIRIS-Layer Tasks for DSM

- **Operator Activation:** Whenever operator instance is needed to be activated in the OSIRIS-layer of a suitable provider, an activation message is sent. This activation messages contains process ID and operator ID. SPM requests the parts of corresponding stream process definition and operator backup information from the replication manager. If the operator instance is executed the first time ever, the operator backup information will be empty and therefore no internal state initialization is needed, otherwise the new operator is initialized with a previously saved operator checkpoint because the activation is actually a re-activation of a previously failed operator instance.

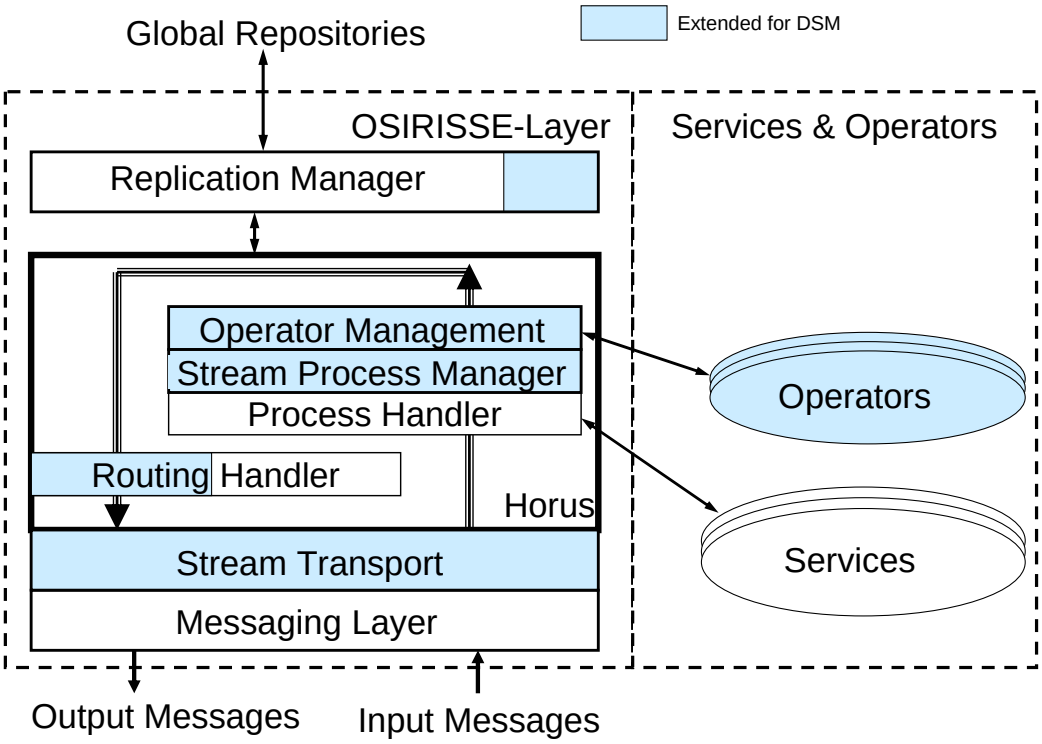


Figure 6.3: The Stream Enabled OSIRIS(SE) Layer

- **Stream Routing:** SPM controls the stream routing locally in peer-to-peer fashion. For each outgoing data stream, suitable service providers for subsequent operators need to be selected. Therefore, SPM retrieves the corresponding subscription and join information from the local replication manager. If a subsequent operator has only a single input stream, this decision can be made locally depending on distributed load information and an activation message is sent to the chosen provider. In case of a subsequent operator with multiple inputs (a join), only one preceding provider, the designated provider, is responsible for making the routing decision. The designated provider is selected by the routing flag of the corresponding TU. The routing decision is propagated to other input providers by updating the join repository via the local replication manager. Providers, for which the routing flag in the corresponding TU is not set, have to wait for an update event on their local replicas. As long as no decision is published, transmission is paused and data elements are stored in transfer state.

- **Operator State Backup:** SPM is also responsible for triggering operator state checkpoints provided by OM, and perform backup of these checkpoints at the backup host. The backup destination is registered in the operator backup repository via the replication manager. An operator state contains internal processing state information, sequence numbers of the last consumed stream elements, sequence numbers of the last produced stream elements, and provider addresses of outgoing streams. Recovery of failed operators can be realized by moving internal states and all input data elements after corresponding input sequence numbers to the backup host, which is able to produce correct outgoing data elements after the corresponding output sequence numbers. An important constraint for the checkpoints of operator states of subsequent operators is to guarantee that checkpoints fit together such that no data elements are missing, this is described as *consistency* of DSM in Section 5.4. Assume two subsequent operators. Then, the last output sequence number in the checkpoint of the first operator has to be the last input sequence number in the checkpoint of the second operator. If the checkpoint of the second operator is later in time, data elements would be missing after recovery. Otherwise, if the checkpoint of the second operator is earlier, data elements would be duplicated after recovery. These duplicates are safely deleted by ST with regard to the deterministic operator model presented in Section 4.1. Details on the implemented reliability algorithm are presented in Chapter 5.

- **Transfer State Trimming:** After a backup of an operator state checkpoint, input elements before checkpoint time are no longer needed for recovery or migration of this operator. OM calls ST to inform the corresponding sender that this operator instance no longer depends on these elements. If ST of the corresponding sender operator receives such acknowledgements from all consuming providers, the referenced data elements in its transfer state are subject for discarding.

- **Migration, Failure and Recovery:** In case of high load, SPM can offload operators by moving operator execution to another suitable provider. In case of a failure, upstream SPM detect it (via ST), and wait for recovery. If there is no recovery within a given time, the designated input SPM uses replicated information from the operator backup repository in order to get information about the responsible backup host. As usually in the HDB infrastructure, this replication is happening in a lazy way. Then, the backup host receives an activation message and instantiate a new operator instance starting from the saved checkpoint. Non-designated input SPM also detect the failure, but need to wait for the update of join information from the replication manager which indicates that the backup provider has successfully recovered the operator instance. Using additional routing information provided in the checkpoint, the newly instantiated operator instance is able to provide the input streams for the consuming operator instances again. In case no suitable backup host is found, SPM can invoke user defined processes, both alternative stream processes or processes, for failure handling.

6.2.4 Stream Process Execution with OSIRIS-SE

As shown in this section, our implemented information management infrastructure OSIRIS-SE will incorporate DSM into process management in peer-to-peer style. Our solution offers great flexibility for healthcare applications. Medical users are able to define individual stream processes for monitoring different patients. Additionally, processes for failure handling, result delivery, critical event handling are supported.

In the activation phase of the stream process, OSIRIS-SE activates all necessary operator instances. This activation is implemented as a traditional process executing single service invocations, like a request for operator activation. If the operator has output data streams, this activation request must contain the routing state to know destinations of output streams. The routing of an operator is performed by the OSIRIS-layer running of the upstream operator instance. In case of multiple upstream operators, a join node, one

of the upstream operators is marked with a routing flag. The OSIRIS-layer hosting the marked operator is responsible for routing and its routing decision is propagated to the other upstream operators via the replication of the join-repository. Secondly, during the running phase of the stream process, OSIRIS-SE monitors all running operator instances and applies the selected reliability algorithms as described in Chapter 5. Finally, during the deactivation phase of a stream process, OSIRIS-SE performs a discrete workflow process to shutdown all running operator instances gracefully. The implementation fulfils the limited-delay, lossless, and intra-stream order preserving reliability levels of Chapter 5.

Example Scenario Revisited: For illustration, we describe how Fred's monitoring applications is using these tasks. Fred's physician has designed the stream process (according to Fig. 3.4) and has also adjusted Fred's operators to his individual parameters (e.g., thresholds for the critical detection). After applying the necessary body sensors to Fred, the stream process is invoked as an execution of an discrete workflow process, by sending activation messages for the sensor operators to Fred's PDA. Beginning with this, all subsequent operators (on Fred's PC and caregiver's PC) are activated in peer-to-peer style. For now, the discrete activation process has terminated successful and all operators are up and continuously processing data streams. In case of critical conditions (e.g., Fred has a heart attack), his physician has designed appropriate processes (e.g., calling the emergency service) which are invoked by the local OSIRIS-layer if necessary. Checkpoints of running operator instances are saved regularly on backup hosts for recovery (e.g., Fred's PC for operators running on Fred's PDA and vice versa). If Fred's PDA fails (e.g., due to empty batteries), operators running on Fred's PDA are migrated to Fred's PC and are initialized with the most recent saved checkpoint. Now, Fred's PC is receiving data directly from the wireless body sensors, because it hosts the acquisition operators. Fred is required to stay near his PC, due to limited transmission range of the body sensors. When Fred is out of range, the OSIRIS-layer on Fred's PC is able to invoke processes to deal with this situation (e.g., inform Fred to come back in range or stop the streaming process gracefully). Because of our peer-to-peer approach local processes may be executed while disconnected from the rest of the OSIRIS-SE network without central control.

7

Evaluation

The experimental evaluation presented in this thesis is twofold. First, we present a real world example application demonstrator. The demonstrator implements a simplified real world telemonitoring scenario by incorporation of a set of real world sensors and mobile devices. This demonstrator is used for illustration purposes on scientific events [BS07], e.g., conferences, to show the functionality of our proposed DSMS infrastructure, to give a hands on impression, and to stimulate discussion.

Second, we are focusing on performance evaluation on both server hardware and mobile devices. For these evaluations, we use real patient sensory data which is processed during evaluations. Moreover, these evaluations cover both the performance during the normal (failure-free) runtime of the DSM system and the performance during the phase of recovering from one or more failures. Additional evaluations are covering performance measurements on stream processes with complex processing graphs including joins, splits, and even feedback cycles where our extended checkpointing algorithms are applied.

7.1 Real-World Example Application Prototype

The demonstration will show an example application within the presented health monitoring scenario (see Section 2.2) based on our OSIRIS-SE infrastructure. Figure 7.1 illustrates the devices included in the demo setup. The smart ECG sensor is continuously acquiring the current heart activity (as ECG signal) of the patient at a rate of 100 samples per second and sending them to an OSIRIS-SE enabled PDA. Additionally, a web cam attached to the OSIRIS-SE enabled laptop is used as a second sensor source producing an im-

age data stream. Moreover, regularly taken measurements with a bluetooth enabled blood pressure sensor device and a bluetooth enabled scale are also forwarded to an OSIRIS-SE enabled PDA. In addition, an bluetooth enabled acceleration sensor applied at the patient's body allows for continuous measurement of motion information. This motion data stream is forwarded to the base station. Laptops and ultra-mobile PCs are used for the base station at the patient's home and the caregiver's server.

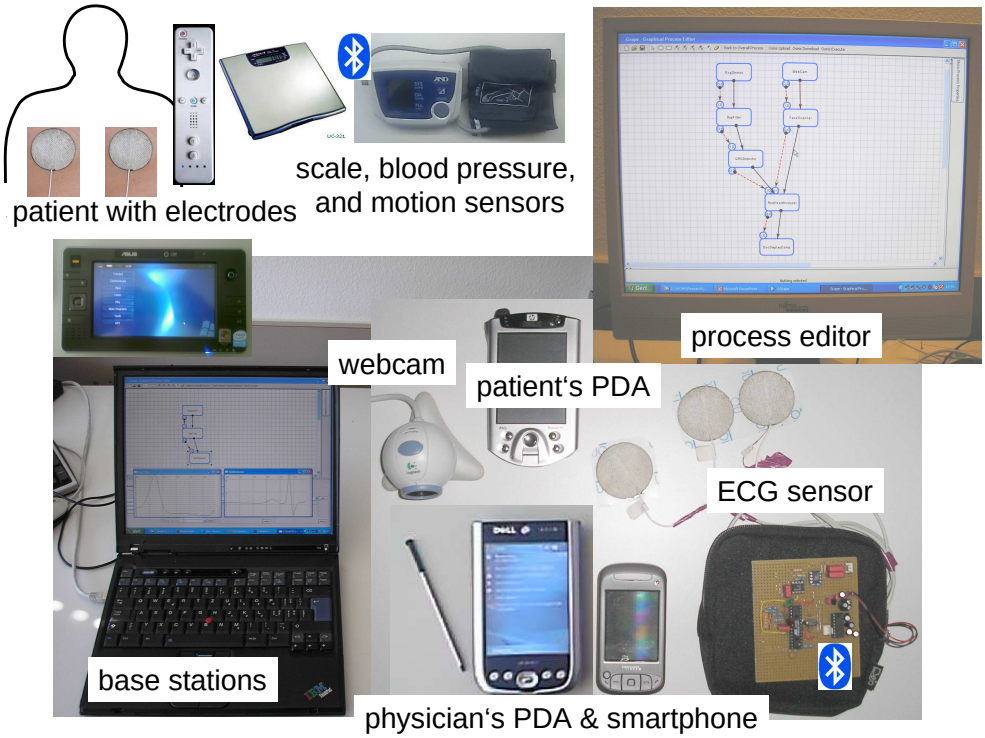


Figure 7.1: Application Prototype Setup

The stream process used by this application is illustrated in Fig. 7.2. This stream process will acquire the ECG signal by the *ECGSensor* operator. Noise is removed from this data stream by applying the *DSPFilter* operator. Medical relevant information about the peaks of the heart activity (so called QRS complexes) is derived by the *QRSDetector* operator. Moreover, the QRSDetector operator also gets input information about the current acceleration level of the patient gathered by the *AccelerationSensor* operator in order to mark detections with high motion as possible prone to movement artifacts. An-

other physiological signal acquired in the demo prototype is blood pressure. For this reason, the *BloodPressureSensor* is continuously querying a wirelessly connected blood pressure device. Whenever the patient is performing a blood pressure measurement, the readings are automatically streamed into the demo. This medical relevant information is combined with context information derived from the web cam data stream. Therefore, the *WebCam* operator is acquiring images of the web cam, which are evaluated by the *FaceScanner* operator. This operator detects faces and calculates their average red value. The *AccelerationSensor* operator delivers necessary movement information about the patient to interpret the physiological signs correctly. The *HealthAnalyzer* operator generates alarm events whenever the combination of red value of the face, the heart activity, the blood pressure, and movement information exceed thresholds given by the caregiver. These alarm events are sent to the *DocDisplayComp* operator at the smartphone of the physician in charge. This example –which can seamlessly be extended by other sensor data sources– nicely illustrates the combination of physiological data streams with contextual data streams. During the demonstration we show that even if the smartphone fails because of battery problems or wireless coverage, the OSIRIS-SE infrastructure is able to seamlessly migrate the *DocDisplayComp* operator to another PDA device without loss of data. Additionally, the demo presents that even multiple failure situations, e.g., also the patient's base station fails at the same time, are handled by OSISIS-SE. Furthermore, the demo will show how to easily define and adapt stream processes with our graphical process editor tool O'GRAPE [WSN+03].

7.2 Performance Evaluations

In this section, we present evaluation results of the different checkpointing strategies presented in Chapter 5 of this thesis. Our reliable DSM is implemented within our information management infrastructure OSIRIS-SE. OSIRIS-SE is fully implemented in Java (see Section 6.1.2). For this reason, each participating node within the OSIRIS-SE infrastructure needs to provide a *Java virtual machine* (JVM). Our evaluation is targeted to measure network transport overhead, CPU load, and memory consumption during the failure-free runtime of a stream process. Due to the requirements demanded by the intended usage for pervasive computing applications including mobile and embedded devices, we consider these resource utilizations as critical. Moreover, we also investigate the behavior during recovery of failures. The performance evaluation is done within two different hardware environments. Firstly, we evaluate the performance within a network of seven mobile de-

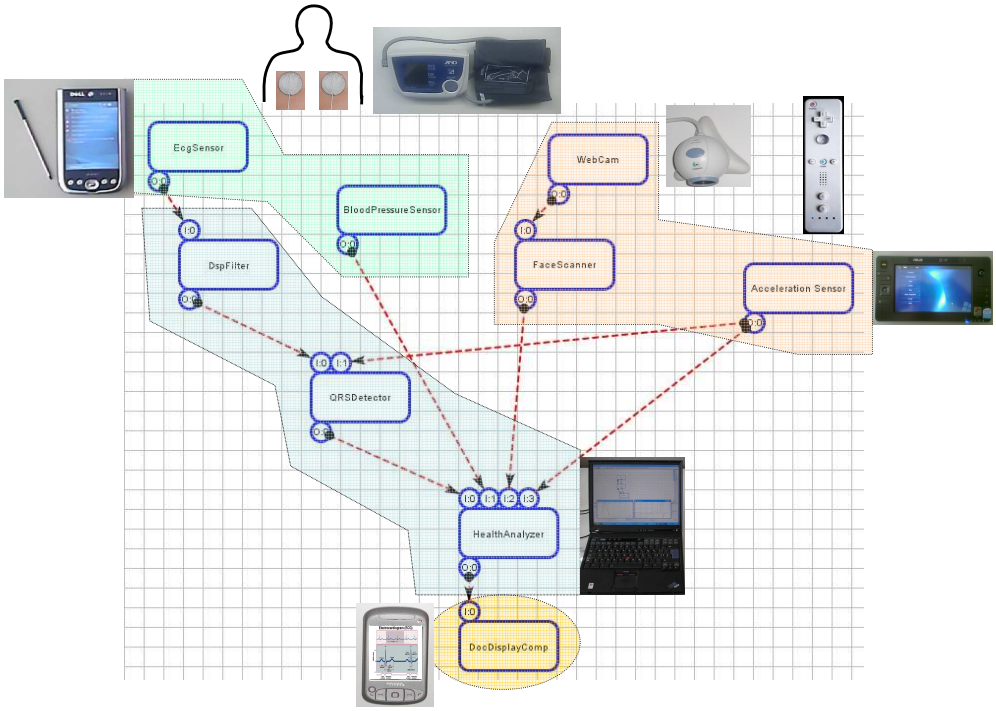


Figure 7.2: Application Prototype Demo Process

vices. Secondly, we perform a deeper evaluation within a network of twelve server-level nodes. On both mobile and stationary platforms, exactly the same Java OSIRIS-SE implementation is executed.

7.2.1 Evaluation Settings

During the evaluation stream processes are executed in four different settings:

1. **Unsafe** stream process execution refers to the execution of a stream process without applying any reliability strategy. In this case, no recovery by operator migration is possible in case of failures. In the unsafe setting, *no operator checkpoints* are scheduled or performed.

2. **Uncoordinated** stream process execution refers to the execution of a stream process with uncoordinated operator checkpointing applied (see

Section 5.9). This case allows for recovery of failures by operator migration. In this setting each operator instance triggers checkpointing locally. In our evaluations, the stream process execution is investigated for different *checkpoint intervals*. Checkpoint intervals are given in number of data stream elements processed between two subsequent checkpoints. For the uncoordinated setting, a fixed checkpoint interval would again cause some form of coordination of checkpoints between the operators. For this reason, checkpointing is not performed precisely the defined number of elements for the checkpoint interval in the uncoordinated setting. At each checkpoint the exact time of the next checkpoint is chosen randomly within an interval from half of the given checkpoint interval to one and a half of the given checkpoint interval.

3. **Coordinated** stream process execution refers to the execution of a stream process with our ECOC operator checkpointing strategy applied (see Section 5.10). This case allows for recovery of failures by operator migration. In the coordinated setting only sensor operators trigger checkpoints. All other operators receive checkpoint requests via data stream connections. For sensor operators, the checkpoints are scheduled according to the given checkpoint interval parameter.

4. **Extended** stream process execution refers to the execution of a stream process with our ECOC operator checkpointing strategy applied (see Section 5.11). This case allows for recovery of failures by operator migration even for stream process topologies that contain cycles in the flow of data stream processing. Also in the extended setting only sensor operators trigger checkpoints and the checkpoints are scheduled according to the given checkpoint interval parameter.

7.2.2 Investigated Parameters

Firstly, the following resource utilizations are measured during the failure-free runtime of different stream processes and while applying the different reliability strategies presented in this thesis.

- *Network transport* overhead is the additional amount of communication data caused due to our reliability strategies. The overhead is measured relative as ratio of bytes needed for sending checkpoint messages to bytes needed for sending data stream elements between running operator instances.

- *CPU load* is the utilization of the CPU of a participating node during the execution of a stream process.

- *Memory consumption* is the additional amount of main memory of a participating node imposed by our reliability measures during the execution of a stream process. Memory consumption is provided by the JVM. For this reason, some deviations will occur in this measurement due to the heuristics of the JVM's garbage collector.

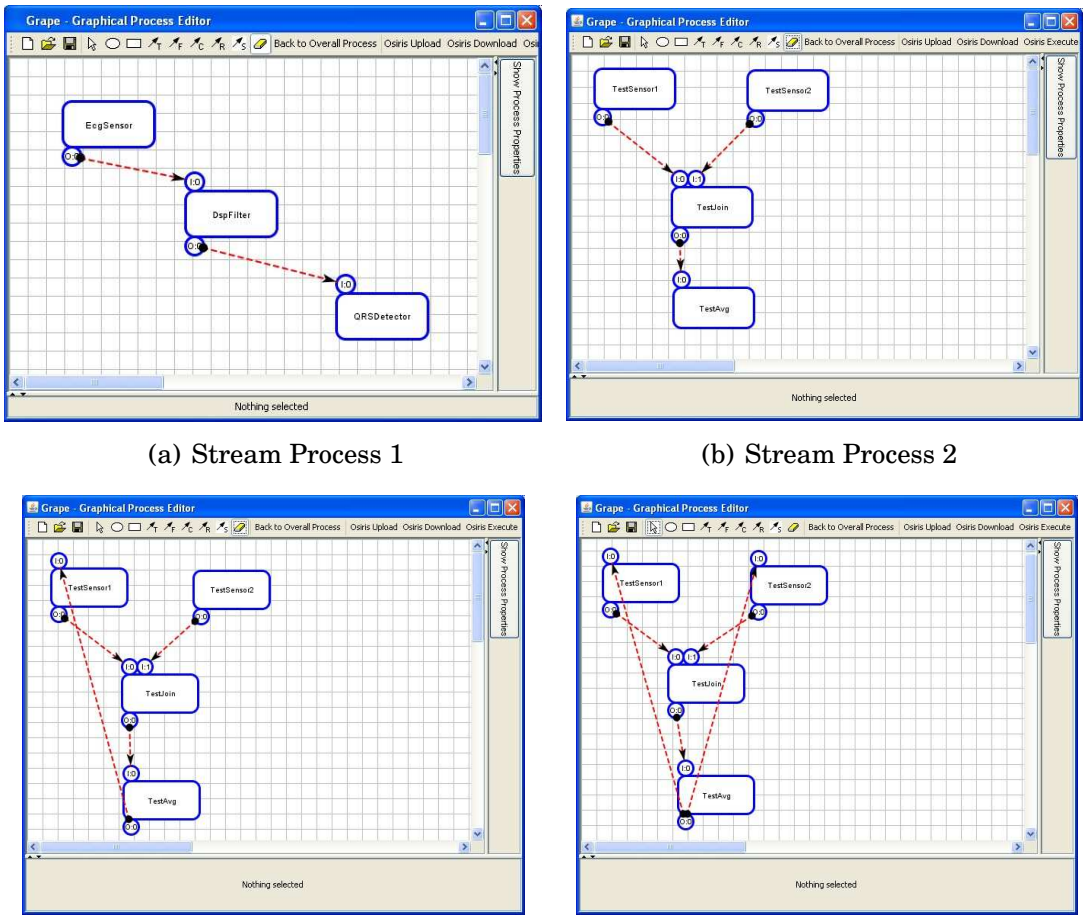
Furthermore, we evaluate the performance of failure handling of our presented reliability strategies during the operator migration phase:

- *Recovery time* is the time τ_r needed for instantiation of a new operator instance and reconstruction of the recent operator state from the checkpoint (see Fig. 5.3).
- *Catchup time* is the time τ_c needed to work off the congestion that has piled up during the time when the failed operator instance was not available (see Fig. 5.3).
- Resource utilization at the recovering node. The CPU and memory utilization is presented as average value during the recovery phase τ_r and the catchup phase τ_c.

7.2.3 Evaluation Stream Processes

For the experiments, the sample stream process depicted in Figure 7.3a is implemented to process real world ECG data within a healthcare application. The *ECGSensor* operator is simulating an sensor for a single-lead human *electrocardiogram* (ECG) by reading data values from a file. The data file contains real world ECG data coming from PhysioNet [Phy]. Each data sample within the file contains one float value for the real-world timestamp of measurement and one float value for the ECG voltage. The *DSPFilter* operator is processing the incoming raw-ECG data stream by applying a FIR filter of fourth order. This processing also removes high frequency distortions, e.g., noise. Finally, the preprocessed ECG data stream is arriving at the *QRSDetector* operator. The QRSDetector implements a single scan QRS-detection and feature extraction algorithm based on [WC79]. The *QRS complex* is the characteristic shape within the ECG which is caused by depolarization of the heart ventricles. Changes in the shape of the QRS complex allow for diagnosis of various diseases of the heart.

The sample stream process depicted in Figure 7.3b is implemented to allow for the analysis of more complex stream processes including a join of two different data streams. This stream process is more artificial than the stream process of Fig. 7.3a. Nevertheless the processing performed by the operators is



(a) Stream Process 1

(b) Stream Process 2

(c) Stream Process 3

(d) Stream Process 4

Figure 7.3: The Evaluation Stream Processes

comparable to data processing needed in various application domains. Again the two sensor operators (*TestSensor1* and *TestSensor2*) are generating sensor data streams containing two float values (timestamp and value) for each data stream element (sample). The *TestJoin* operator is performing as integration of the sum of both data streams of a sliding window in time. Finally, the *TestAvg* operator is performing an average over a sliding window in time on its incoming data stream. Due to the sliding window operations performed by TestJoin and TestAvg the internal state of the operators is bigger than in the first stream process.

The sample stream processes depicted in Figure 7.3c and Figure 7.3d are extensions of the stream process in Figure 7.3b. Additional, feedback cycles from the *TestAvg* to one or both *TestSensor* operators allow to influence the

sensor processing. In these sample processes, generated sensor values are attenuated if the result of *TestAvg* exceeds a certain threshold.

7.2.4 Procedure of Evaluation

For both mobile and stationary environments a performance evaluation is presented in the next sections. The performance evaluation investigates two different situations of a stream process.

Firstly, the *failure-free runtime* is evaluated. In this case, the stream process is up and running without any failure situation like crashed operator providers or network disconnections. In order to avoid any disturbances, the stream process is executed in a way that each operator instance is hosted by a different provider node. Moreover, each operator instance has a dedicated backup node which is not performing other tasks. The actual selection of provider nodes and backup nodes is done by the OSIRIS-SE infrastructure based on current load situations. During the experiment the stream process is executed for each setting (unsafe, uncoordinated, coordinated, extended) in combination with the different backup intervals (500, 1000, 1500, 2000, 2500, 3000) for a duration of 400 seconds and averaged logging statistics are collected. Of course, for the unsafe setting the checkpoint interval is not applicable. Therefore, there are 19 combinations of settings and backup intervals. In order to avoid the influence random disturbances (e.g., operating system tasks) the execution of stream processes is repeated several times. The presented results are aggregated over the execution time and the number of repetitions. Moreover, only one stream process is executed at the same time. For the mobile environment the measurements were repeated 5 times and for the stationary environment 10 times. Therefore in total 95 stream processes were executed for the mobile environment and 190 for the server environment for each of the evaluated stream processes. In order to avoid interference of concurrent stream processes only one stream process was executed at the same time.

Secondly, the *failure handling* is evaluated. In this experiment, the stream process is set up and running again without failure situation for a duration of 150 seconds. After that an operator failure is explicitly triggered. In order to avoid time synchronization effects, a random delay of between 0 and 50 seconds is introduced before the actual failure triggering. As for the failure-free measurement, this measurement is repeated 5 times for the mobile environment and 10 times for the stationary environment. Also in this situation, only one stream process executed at the same time. In order to compare the different strategies, the measurements are performed for each setting (uncoordinated, coordinated, and extended) in combination with the different check-

point intervals (500, 1000, 1500, 2000, 2500, 3000). Of course, performing this measurement with the unsafe setting is not possible. For this reason, there are 18 combinations in this experiment. This leads to 180 stream process executions in the server environment and 90 stream process executions in the mobile environment for each of the evaluated stream processes.

7.2.5 Performance Evaluations on Mobile Computers

This experimental setup consists of four *Dell Axim X51v* PDAs and three *HTC TyTN* smartphones (see Fig. 7.4). The PDAs are equipped with an I*ntel XScale* processor with 624 MHz and 64MB of main memory. The smartphones have a *Samsung SC32442A* processor with 400 MHz and also 64MB of main memory. Both device types are running on the *Windows Mobile 5* operating system. Each mobile device has a local OSIRIS-SE software layer hosted by the J9 Java virtual machine and all devices are connected via wireless connection (W-LAN). The J9 JVM is provided by IBM within the *WebSphere Everyplace Micro Environment Runtime* [WeM]. An additional laptop computer is used to host the global repositories. Due to the Peer-to-Peer nature of stream process execution with OSIRIS-SE, the laptop is not directly involved in stream process execution. The laptop computer is not hosting any operator instances. The stream processes within the mobile environments are executed with a rate of 30 data stream elements per second produced by each sensor operator.

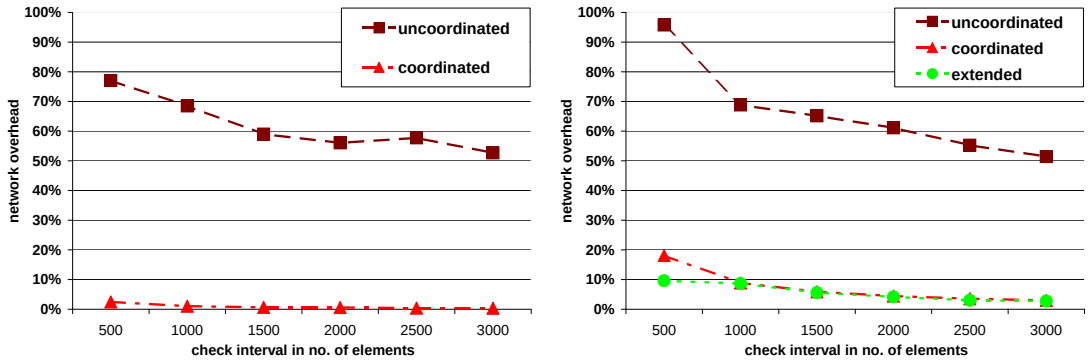
Node	Operators (Stream Process 1)	Operators (Stream Process 2)
PDA 1	ECGSensor	TestSensor1
PDA 2	ECGSensor	TestSensor2
PDA 3	DSPFilter	TestJoin
PDA 4	DSPFilter	TestJoin
Smartphone 1	QRSDetection	TestAvg
Smartphone 2	QRSDetection	TestAvg
Smartphone 3	QRSDetection	TestAvg

Table 7.1: Operator Provider in Mobile Environment

Table 7.1 illustrates which operators are available at which nodes in the experiments. Since all operators are available at more than one node in the OSIRIS-SE network, the infrastructure is able to select one node as operator provider and another node as backup provider for each operator instance. The only exception is for Stream Process 2 where the backup provider for both TestSensor1 and TestSensor2 is the additional laptop computer. However, this has no effect on the measurement because only the operator providers

Figure 7.4: The setting of the mobile evaluation

are evaluated. There is also no affection on the failure measurement because no failures are triggered on either TestSensor1 or TestSensor2.



(a) Stream Process 1

(b) Stream Process 2

Figure 7.5: Network overhead during failure-free runtime in mobile setting

Data Stream	Check Interval	Uncoordinated	Coordinated
7.27 KB/s	500	5.60 KB/s	0.18 KB/s
7.27 KB/s	1000	4.98 KB/s	0.08 KB/s
7.27 KB/s	1500	4.29 KB/s	0.05 KB/s
7.27 KB/s	2000	4.08 KB/s	0.04 KB/s
7.27 KB/s	2500	4.20 KB/s	0.03 KB/s
7.27 KB/s	3000	3.83 KB/s	0.03 KB/s

Table 7.2: Transfer Rates for Stream Process 1

Data Stream	Check Interval	Uncoordinated	Coordinated	Extended
8.04 KB/s	500	7.70 KB/s	1.45 KB/s	0.77 KB/s
8.04 KB/s	1000	5.53 KB/s	0.71 KB/s	0.70 KB/s
8.04 KB/s	1500	5.24 KB/s	0.47 KB/s	0.45 KB/s
8.04 KB/s	2000	4.91 KB/s	0.35 KB/s	0.33 KB/s
8.04 KB/s	2500	4.44 KB/s	0.28 KB/s	0.24 KB/s
8.04 KB/s	3000	4.14 KB/s	0.24 KB/s	0.22 KB/s

Table 7.3: Transfer Rates for Stream Process 2

Fig. 7.5 illustrates the results for the network overhead during failure-free runtime of a stream process. Fig. 7.5 is based on data coming from Table 7.2 and Table 7.3. These tables contain the transfer rates for the stream transport and the additional checkpointing as absolute numbers. The first column indicates the sum of average transfer rates of all data streams sent within the stream process. The second column indicates the checkpoint interval and the remaining columns contain the average transfer rates of checkpointing messages needed for the different checkpointing algorithms. Comparing the coordinated and uncoordinated setting, we see that our ECOC approach is *significantly reducing* the overhead for checkpointing due to transport of checkpoint messages from operator provider to backup provider.

Comparing Stream Process 1 and Stream Process 2, we see a slightly higher network overhead for the coordinated setting. This is due to larger operator states in Stream Process 2.

At the shortest checkpoint interval of 500 the uncoordinated setting shows a significant increase of the network overhead for Stream Process 2. This corresponds to the qualitative analysis of the checkpoint overhead in Section 5.8.2. When the minimal checkpoint interval (MCI) is reached the transfer state will no more reduce by reducing the checkpoint interval.

Analyzing the performance of the extended ECOC setting, we see a significant improvement (from 18% down to 9%) in Stream Process 2 compared to the coordinated setting with 500 checkpoint interval. For longer check-

point intervals no improvement is measured. The reason for that is explained by Figure 7.6. Fig. 7.6 show the percentage of checkpoints that have been extended during stream process execution. For 500 the extended ECOC strategy has extended 20% of the checkpoints. For longer checkpoint intervals no extension was performed because extensions are only performed if two subsequent pending checkpoints are relatively close in stream-time. The chance for this is decreasing by increased checkpoint intervals. The extended ECOC setting has not been applied to Stream Process 1 for the mobile environment. Since there is no expected improvement due to the extended setting for the simple three-staged topology of Stream Process 1 this measurement has been omitted. In order to empirically proof this statement, the measurement has been conducted for Stream Process 1 within stationary environment (see Section 7.2.6).

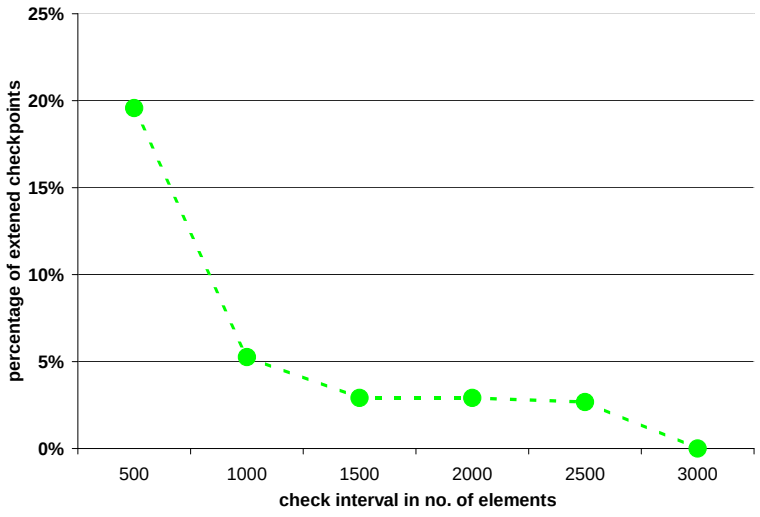
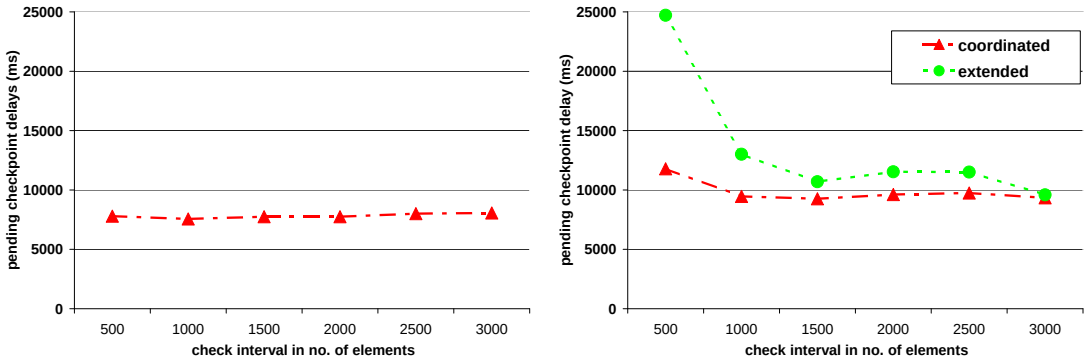


Figure 7.6: The ratio of extended backups for Stream Process 2

Fig. 7.7 illustrates the average delays for pending checkpoints until they become permanent checkpoints and are transferred to the backup provider. These delays are only existent in the coordinated and extended settings. In the uncoordinated setting each checkpoint is immediately stored at the backup provider. For the coordinated setting the delays are independent from the checkpoint interval. For the 500 checkpoint interval of Stream Process 2, there is a significant increase of the pending checkpoint delay while for higher checkpoint intervals it is only slightly increased. The explanation for

this is that, the extension of a pending checkpoint introduces an additional delay and for higher checkpoint intervals less checkpoints are extended. In general, the pending checkpoint delays do not disturb stream processing and also do not degrade the failure handling performance as results will show in the remainder of this section.



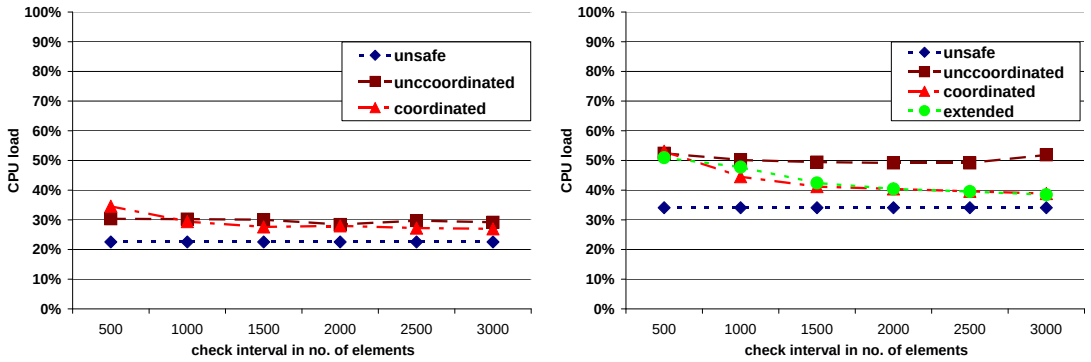
(a) Stream Process 1

(b) Stream Process 2

Figure 7.7: Delay of pending checkpoints during failure-free runtime

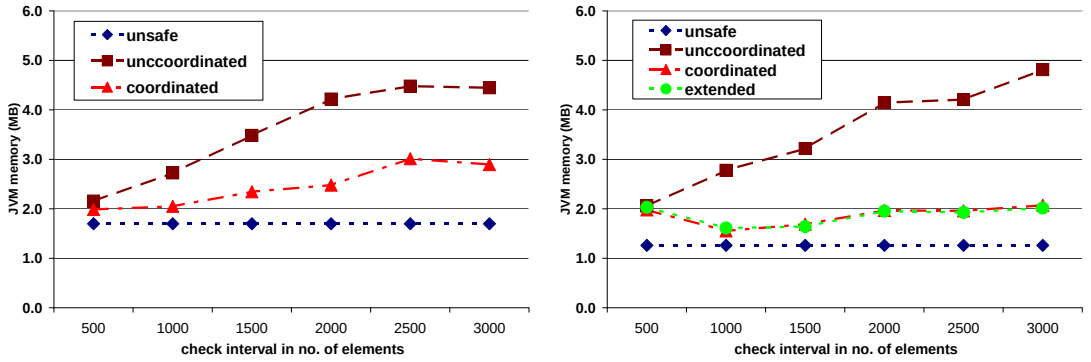
Fig. 7.8 illustrates the CPU load of the different reliability strategies. Compared to the unsafe setting where no reliability is applied to stream processing, the overhead of CPU load imposed by reliability is reasonable. For Stream Process 1 which has a simple topology this overhead is less than in the more complex Stream Process 2. Another result indicated by Fig. 7.8 is that the coordinated und extended settings are slightly less CPU demanding than the uncoordinated setting. This effect does not appear in the same measurement for the stationary setting (see Fig. 7.19). An explanation for this could be the fact that the uncoordinated setting is sending more data for checkpointing. On a mobile device with a wireless module this task is more CPU intensive compared to the server environment.

Fig. 7.9 illustrates the average JVM memory consumption of a node during stream process execution. Compared to the unsafe setting in which no reliability is applied to stream processing, the overhead of memory imposed by reliability is reasonable. In particular the coordinated and extended settings show only slightly higher memory demand. The significantly higher memory demand for the uncoordinated settings comes from the need to checkpoint the transfer state. During checkpointing of the transfer state, larger data structures in memory are needed to send the larger checkpoint messages. This fact is also pointed out by increasing memory overhead for longer checkpoint intervals which lead to larger checkpoint messages.



(a) Stream Process 1 (b) Stream Process 2

Figure 7.8: CPU load during failure-free runtime

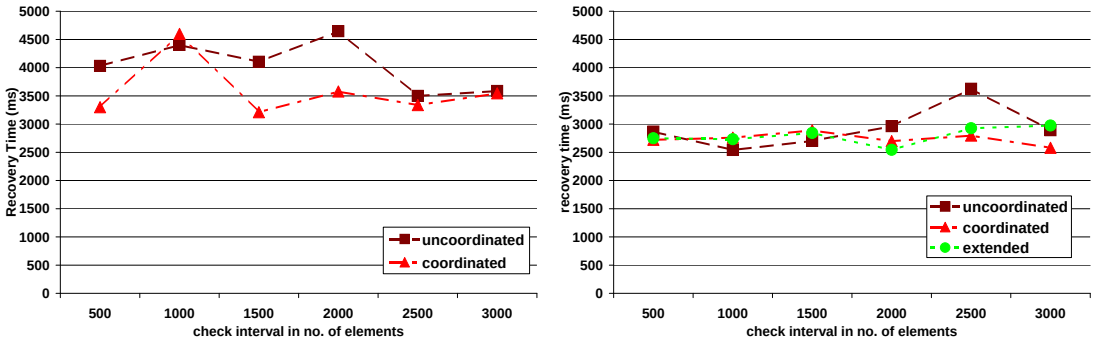


(a) Stream Process 1 (b) Stream Process 2

Figure 7.9: JVM memory consumption during failure-free runtime

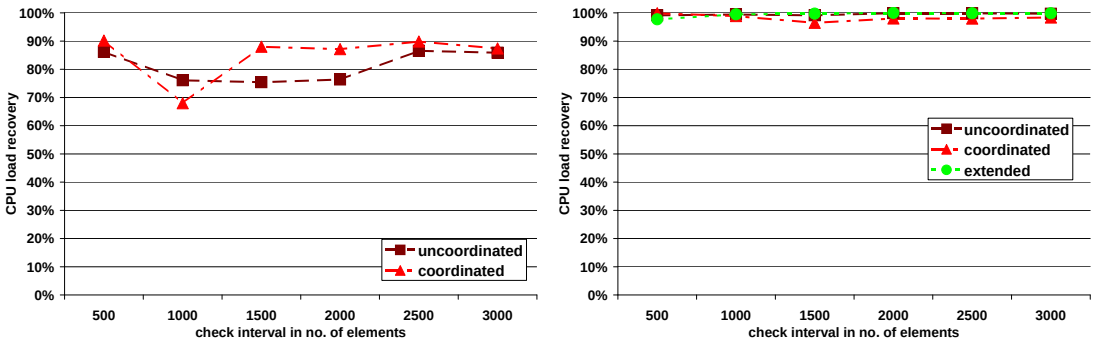
The failure handling in the mobile environment was evaluated by triggering an operator failure of the QRSDetector operator for Stream Process 1 and of the TestAvg operator for Stream Process 2. Fig 7.10 illustrates the time for recovery τ_r of a failed operator instance. There is no significant difference between the different strategy approaches. There is a slightly shorter recovery time for Stream Process 2 which is connected to the higher CPU utilization during recovery in Stream Process 2 (see Fig. 7.11). Whereas JVM memory consumption (see Fig. 7.12) is almost constant in all settings.

Fig 7.13 illustrates the time for catchup τ_c of a failed operator instance. During the catchup phase, a newly recovered operator instance is working off the congestion that was caused during the time of failure. There are no



(a) Stream Process 1 (b) Stream Process 2

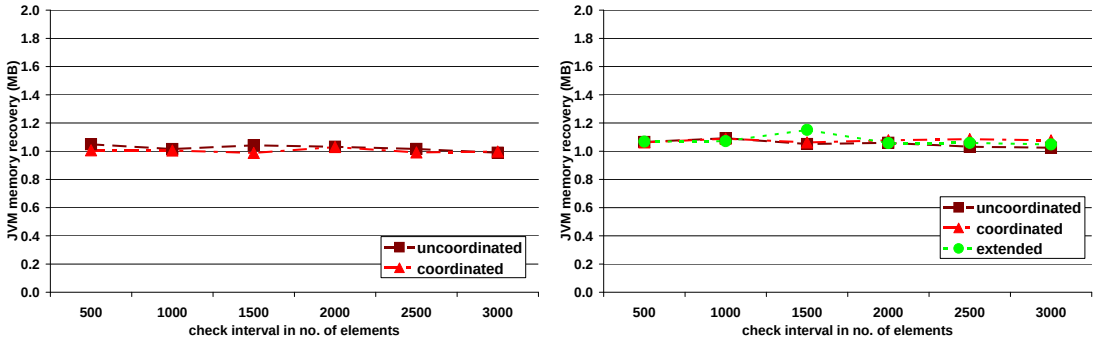
Figure 7.10: Recovery Time



(a) Stream Process 1 (b) Stream Process 2

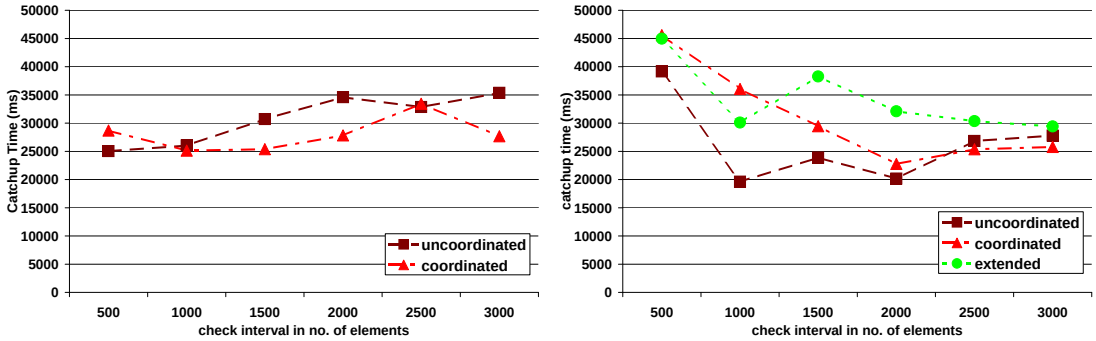
Figure 7.11: CPU load during Recovery Time

significant differences between the different reliability algorithms. Also CPU load (see Fig. 7.14) and JVM memory consumption (see Fig 7.15) are within reasonable variations. Only for higher checkpoint interval an additional JVM memory overhead is caused for Stream Process 1 and the uncoordinated setting. This behavior is similar to the failure-free evaluation results.



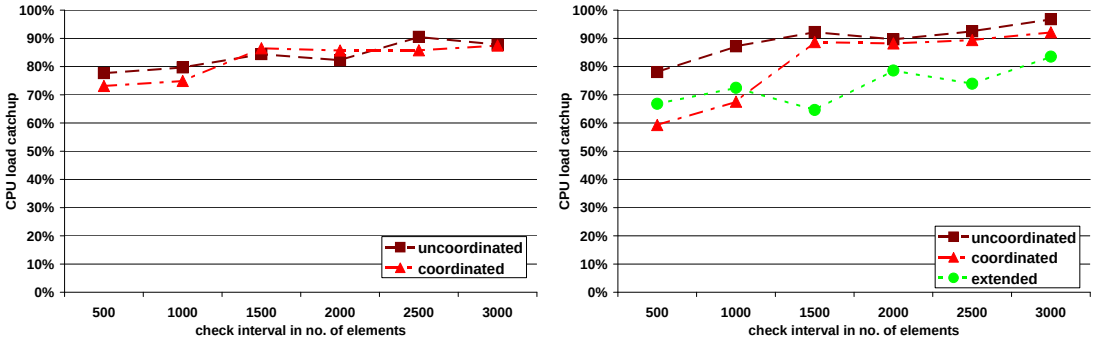
(a) Stream Process 1 (b) Stream Process 2

Figure 7.12: JVM memory consumption during Recovery Time



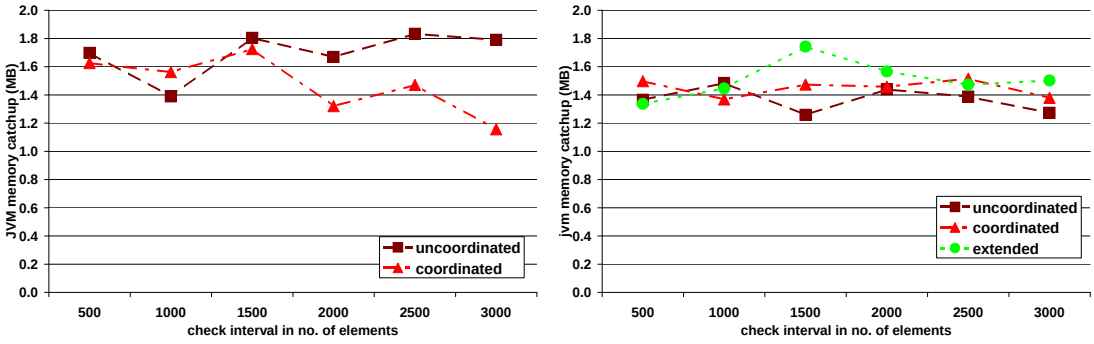
(a) Stream Process 1 (b) Stream Process 2

Figure 7.13: Catchup Time



(a) Stream Process 1 (b) Stream Process 2

Figure 7.14: CPU load during Catchup Time



(a) Stream Process 1 (b) Stream Process 2

Figure 7.15: JVM memory consumption during Catchup Time

7.2.6 Performance Evaluations on Stationary Devices

The experimental setup consists of a network of twelve server nodes. Each node has an Intel Xeon CPU with 3.2GHz, 2GB of main memory, and is running on the *Windows Server 2003* operating system. One node is dedicated to host the global repositories. The others are operator and backup providers. All nodes are equipped with a local OSIRIS-SE software layer which is hosted by a *Sun J2SE1.6* JVM [J2S] and thus are able to run the evaluation stream processes. As for the mobile environment, a node is not operator provider and backup provider at the same time and only the operator providers are evaluated. All nodes are connected via a reliable Gigabit Ethernet connection. The stream processes within the stationary environments are executed with a rate of 200 data stream elements per second produced by each sensor operator.

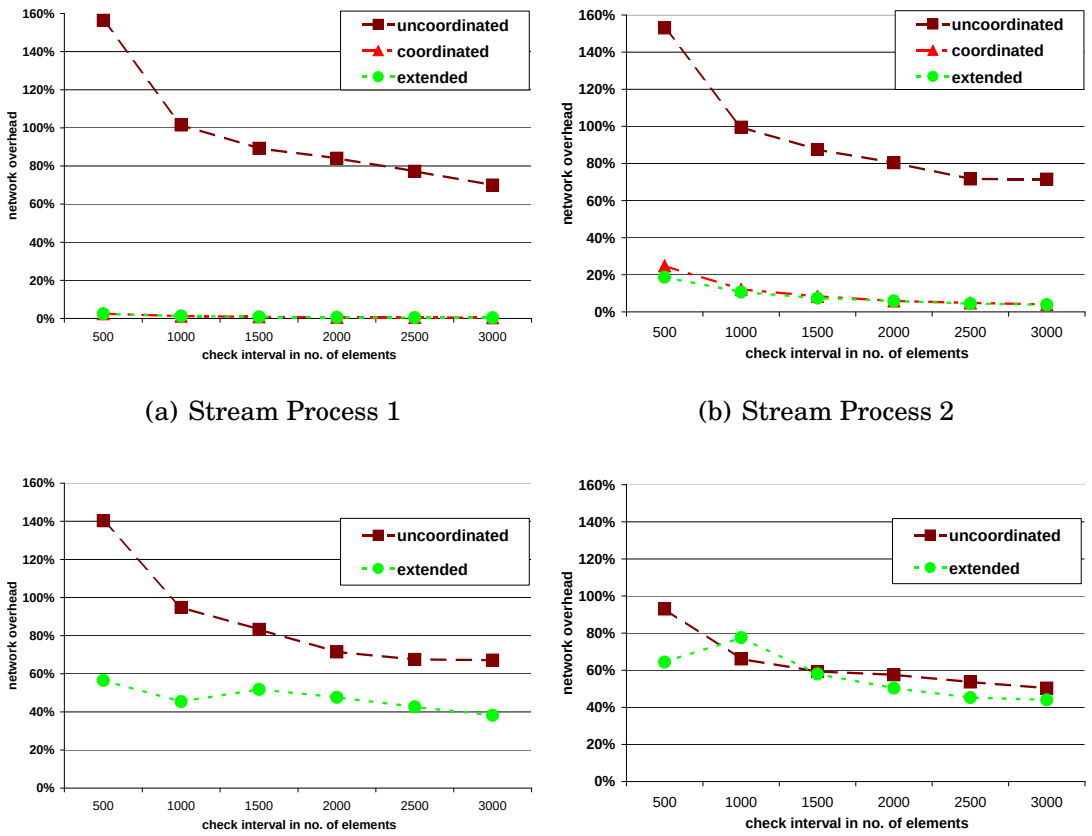
Node	Operators (Stream Process 1)	Operators (Stream Process 2,3,4)
Server 1	ECGSensor	TestSensor1
Server 2	ECGSensor	TestSensor1
Server 3	ECGSensor	TestSensor2
Server 4	ECGSensor	TestSensor2
Server 5	DSPFilter	TestJoin
Server 6	DSPFilter	TestJoin
Server 7	DSPFilter	TestJoin
Server 8	DSPFilter	TestJoin
Server 9	QRSDetection	TestAvg
Server 10	QRSDetection	TestAvg
Server 11	QRSDetection	TestAvg

Table 7.4: Operator Provider in Server Environment

Table 7.4 illustrates which operators are available at the server nodes in the experiments. Since all operators are available at more than one node in the OSIRIS-SE network, the infrastructure is able to select one node as operator provider and another node as backup provider for each operator instance.

Data Stream	Check Interval	Uncoordinated	Coordinated	Extended
29.02 KB/s	500	45.40 KB/s	0.72 KB/s	0.73 KB/s
29.02 KB/s	1000	29.46 KB/s	0.35 KB/s	0.37 KB/s
29.02 KB/s	1500	25.90 KB/s	0.25 KB/s	0.24 KB/s
29.02 KB/s	2000	24.39 KB/s	0.18 KB/s	0.17 KB/s
29.02 KB/s	2500	22.41 KB/s	0.14 KB/s	0.13 KB/s
29.02 KB/s	3000	20.31 KB/s	0.11 KB/s	0.11 KB/s

Table 7.5: Transfer Rates for Stream Process 1



(a) Stream Process 1

(b) Stream Process 2

(c) Stream Process 3

(d) Stream Process 4

Figure 7.16: Network overhead during failure-free runtime in stationary setting

Data Stream	Check Interval	Uncoordinated	Coordinated	Extended
47.80 KB/s	500	73.15 KB/s	11.88 KB/s	8.95 KB/s
47.80 KB/s	1000	47.57 KB/s	5.82 KB/s	5.13 KB/s
47.80 KB/s	1500	41.82 KB/s	3.99 KB/s	3.55 KB/s
47.80 KB/s	2000	38.40 KB/s	2.82 KB/s	2.80 KB/s
47.80 KB/s	2500	34.25 KB/s	2.31 KB/s	2.15 KB/s
47.80 KB/s	3000	34.20 KB/s	1.90 KB/s	1.86 KB/s

Table 7.6: Transfer Rates for Stream Process 2

Also in the server environment, the network overhead during failure-free runtime of a stream process is significantly reduced by applying the coordinated and the extended reliability algorithm compared to the uncoordinated algorithm (see Fig. 7.16). Fig. 7.16 is based on data coming from Table 7.5,

Data Stream	Check Interval	Uncoordinated	Extended
59.25 KB/s	500	83.10 KB/s	33.50 KB/s
59.25 KB/s	1000	56.16 KB/s	26.89 KB/s
59.25 KB/s	1500	49.36 KB/s	30.70 KB/s
59.25 KB/s	2000	42.33 KB/s	28.25 KB/s
59.25 KB/s	2500	39.98 KB/s	25.26 KB/s
59.25 KB/s	3000	39.77 KB/s	22.62 KB/s

Table 7.7: Transfer Rates for Stream Process 3

Data Stream	Check Interval	Uncoordinated	Extended
75.21 KB/s	500	69.95 KB/s	48.44 KB/s
75.21 KB/s	1000	49.69 KB/s	58.34 KB/s
75.21 KB/s	1500	44.61 KB/s	43.64 KB/s
75.21 KB/s	2000	43.28 KB/s	37.88 KB/s
75.21 KB/s	2500	40.37 KB/s	34.07 KB/s
75.21 KB/s	3000	37.89 KB/s	33.10 KB/s

Table 7.8: Transfer Rates for Stream Process 4

Table 7.6, Table 7.7, and Table 7.8. These tables contain the transfer rates for the stream transport and the additional checkpointing as absolute numbers. The first column indicates the sum of average transfer rates of all data streams sent within the stream process. The second column indicates the checkpoint interval and the remaining columns contain the average transfer rates of checkpointing messages needed for the different checkpointing algorithms.

As already shown in the mobile environment, the network overhead of the uncoordinated setting is exponentially increasing for shorter checkpoint intervals. This corresponds to the qualitative analysis of the checkpoint overhead in Section 5.8.2. When the minimal checkpoint interval (MCI) is reached the transfer state will no more reduce by reducing the checkpoint interval. This is caused by the inherent delay between producing and consuming data stream elements. During the delay the producer may already produce additional elements before the consumer has received the previous ones. As a result of this, some data stream elements are participate in multiple checkpoint messages as part of the transfer state. If this happens, the network overhead can reach even more than 100 percent.

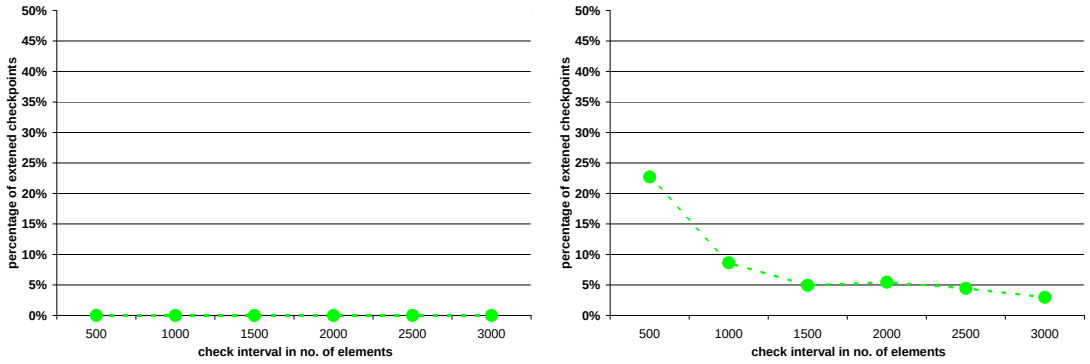
When further analyzing the uncoordinated setting for the various stream processes, we see that for the Stream Process 3 and 4 (the ones with cycles in the data flow) the network overhead is reduced compared to Stream Process 2 (without cycles). The reason for this is that the network overhead is measured

as ratio of bytes needed for sending of checkpoint messages to the number bytes needed for sending of data stream elements. In Stream Process 3 and 4 the bytes needed for checkpoint messages have not that much increased as the additional bytes needed for sending data stream elements along the new feedback data streams (see Table 7.7 and Table 7.8). This results in less network overhead for a more complex stream process in the uncoordinated setting.

Analyzing the performance of the extended ECOC setting, we see no improvement in Stream Process 1 compared to the coordinated setting. The reason for that is explained by Figure 7.17. Fig. 7.17 show the percentage of checkpoints that have been extended during stream process execution. For the simple Stream Process 1, no checkpoints were extended because there are no joins or cycles in this stream process graph.

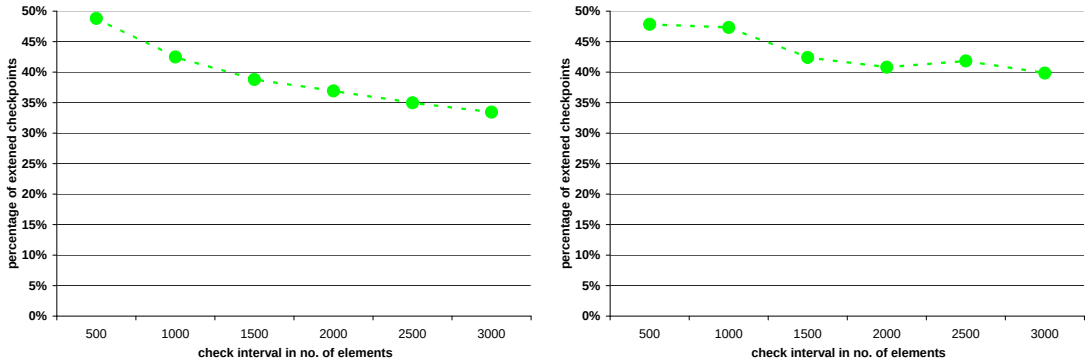
For the more complex Stream Process 2 which also includes a join of two data streams, the extended ECOC setting provides an additional reduction of the network overhead (from 24% down to 18%) for the 500 checkpoint interval. For longer checkpoint intervals, the improvement is disappearing because of less checkpoints being extended (see Fig. 7.17). For the 500 checkpoint interval, the extended ECOC strategy has extended around 23% of the checkpoints. For longer checkpoint intervals, less extensions were performed because extensions are only performed if two subsequent pending checkpoints are relatively close in stream-time. The probability for this is decreasing by increased checkpoint intervals.

For the stream processes with cycles (3 and 4), the coordinated setting is not applicable because checkpoint request would be cascaded in the cycle endlessly. Still our extended ECOC approach overcomes this flaw and is able to deal with cycles in the stream process graph. The extended setting shows a significant reduction of the network overhead in Stream Process 3 (one cycle). For Stream Process 4 (two cycles), there is only a significant reduction for the shortest checkpoint interval. For longer checkpoint intervals the network overhead is becoming comparable to the uncoordinated approach. Fig 7.17 shows an much higher extension rate for Stream Process 3 and 4 compared to Stream Process 1 and 2. The cycles within the stream process graph (3 and 4) require that within cascading the checkpoint request at least one operator instance within the cycle has to extend the checkpoint. The extension due to the cycle is applied in every case and not only when checkpoint requests follow short in time. This is the reason for the decline of the extended ECOC performance with respect to network overhead in stream processes with cycles. Nevertheless for a single cycle within a stream process, the extended setting still significantly outperforms the uncoordinated setting with respect to network overhead.



(a) Stream Process 1

(b) Stream Process 2

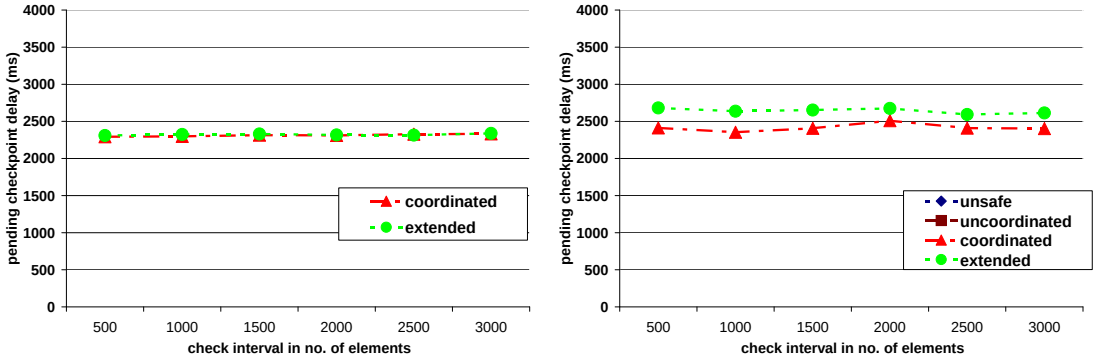


(c) Stream Process 3

(d) Stream Process 4

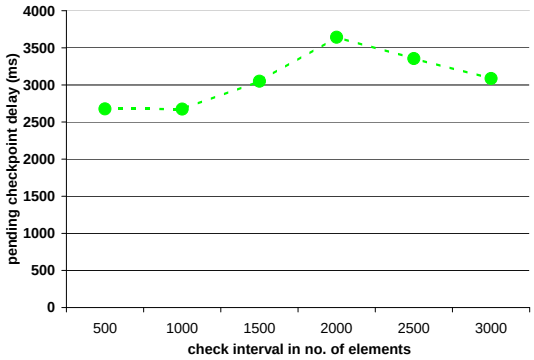
Figure 7.17: Ratio of extended checkpoints during failure-free runtime

Fig. 7.18 illustrates the average delay of a pending checkpoint until it gets a permanent checkpoint. This delay is caused by cascading checkpoint request following the data stream. Checkpoint request are acknowledged immediately at the last operator instance (which is only propagating data to outside world systems) in the data flow or when a cycle of request is detected. Due to this fact, pending checkpoints only occur in the coordinated and extended setting. For Stream Process 2, we see a slightly increased delay for the extended setting which results from the fact that an extended checkpoint is longer delayed. For Stream Process 3, we see even longer delays. For Stream Process 4, the situation changes because due to the two cycles more checkpoint requests are detected as cyclic and immediately trigger a permanent checkpoint. This also explains the increased network overhead for the extended setting when applied to this stream process.

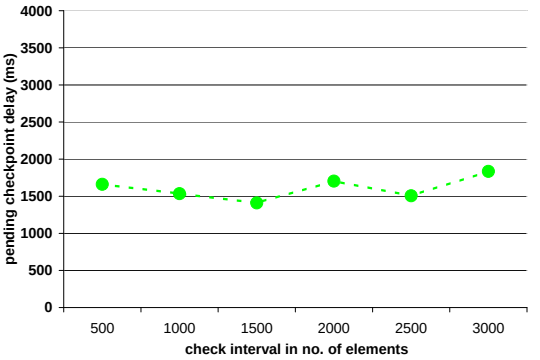


(a) Stream Process 1

(b) Stream Process 2



(c) Stream Process 3

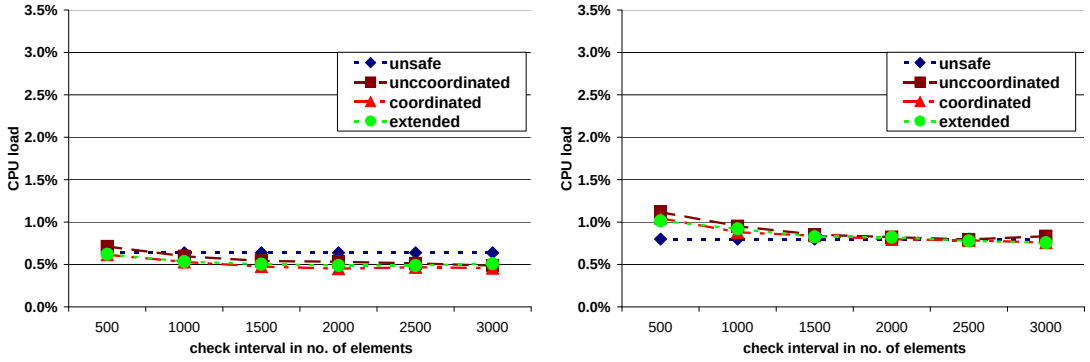


(d) Stream Process 4

Figure 7.18: Delay of pending checkpoints during failure-free runtime

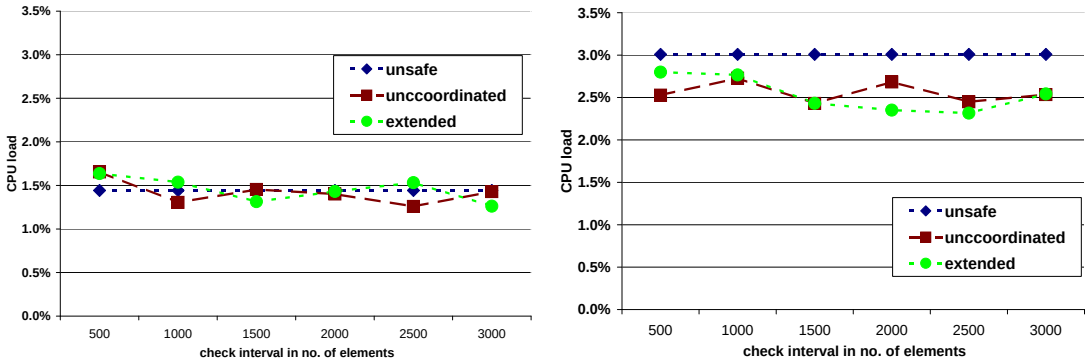
Fig. 7.19 illustrates the CPU load of the different reliability strategies. The experiments show that there is no measurable CPU overhead due to reliability algorithms when comparing to the unsafe setting where no reliability is applied to stream processing. In general, CPU utilization increases with more complex stream process topology. In some experiments, the unsafe setting is slightly more CPU demanding compared the three reliable settings. Explanations for this implausibility are error of measurement and that the higher network utilization in the reliable settings may result in idle CPU cycles due to network access.

Fig. 7.20 illustrates the average JVM memory consumption of a node during stream process execution. Compared to the unsafe setting where no reliability is applied to stream processing, the overhead of memory imposed by reliability is reasonable. In particular the coordinated and extended setting show only slightly higher memory demand for non-cyclic stream processes (1



(a) Stream Process 1

(b) Stream Process 2



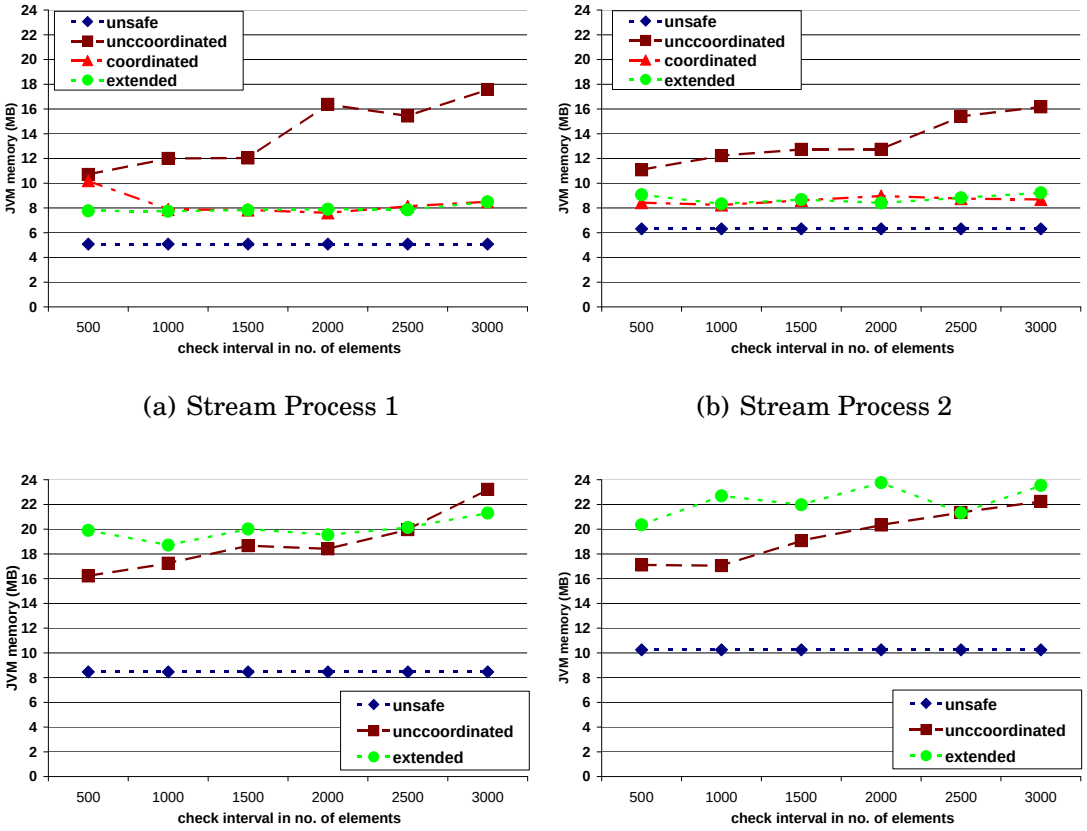
(c) Stream Process 3

(d) Stream Process 4

Figure 7.19: CPU load during failure-free runtime

and 2). The uncoordinated setting requires significantly more memory for non-cyclic stream processes (1 and 2). This significant higher memory demand for the uncoordinated settings comes from the need to checkpoint the transfer state and therefore storing larger data structures in memory. For the cyclic stream processes (3 and 4) the extended setting becomes more memory demanding due to more extended checkpoints. Still, the performance is comparable to the uncoordinated setting.

The failure handling in the stationary environment was evaluated for single and multiple failure situations. For the single failure situation an operator failure of the QRSDetector operator for Stream Process 1 and of the TestAvg operator for Stream Process 2,3,4 has been triggered. For the multi failure situation the DSPFilter operator of Stream Process 1 and the TestJoin operator of Stream Process 2,3,4 failed in addition. Fig 7.21 illustrates the time for recovery τ_r of a failed operator instance for the single failure scenario and



(a) Stream Process 1

(b) Stream Process 2

(c) Stream Process 3

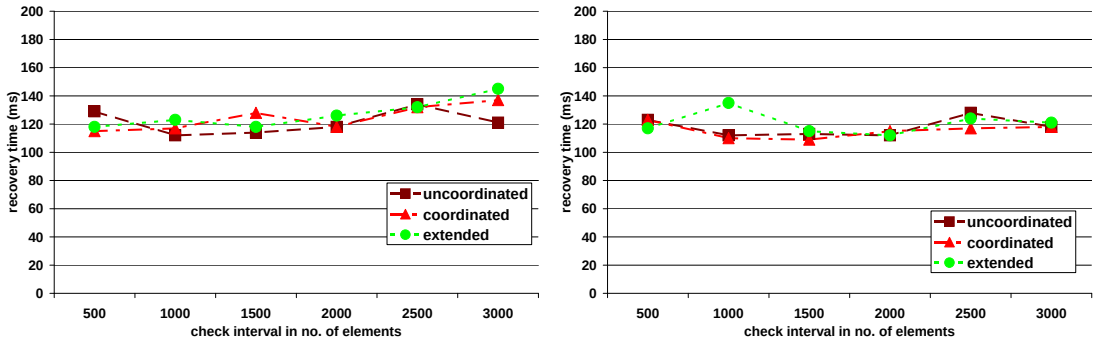
(d) Stream Process 4

Figure 7.20: JVM memory consumption during failure-free runtime

Fig. 7.22 for the multiple failure scenario. There are no significant differences between the different reliability approaches.

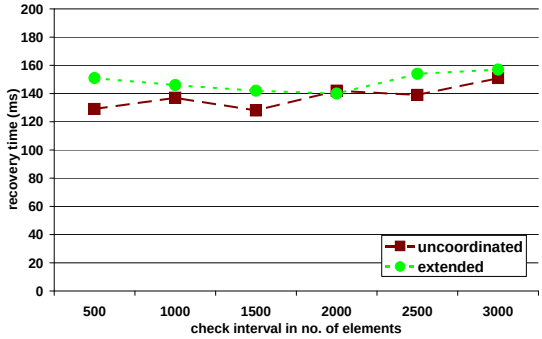
Also CPU utilization (see Fig. 7.23 and Fig. 7.23) and JVM memory consumption (see Fig. 7.25 and Fig. 7.26) have no significant variation between the different reliability settings.

Fig 7.27 illustrates the time for catchup τ_c of a failed operator instance for the single failure scenario and Fig. 7.28 for the multiple failure scenario. There are no significant differences between the different strategy approaches. Except the fact that for Stream Process 4 (containing two feedback cycles), the catchup time of the extended approach is increased. More intensive checkpoint extension and loop detection seems to retard the catchup process in this case. This issue may be caused by having less permanent checkpoints compared to the uncoordinated case because the ratio of extended

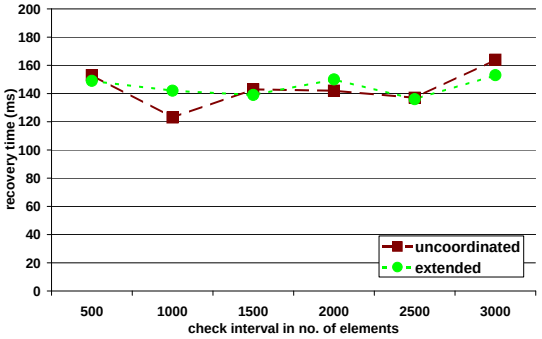


(a) Stream Process 1

(b) Stream Process 2



(c) Stream Process 3

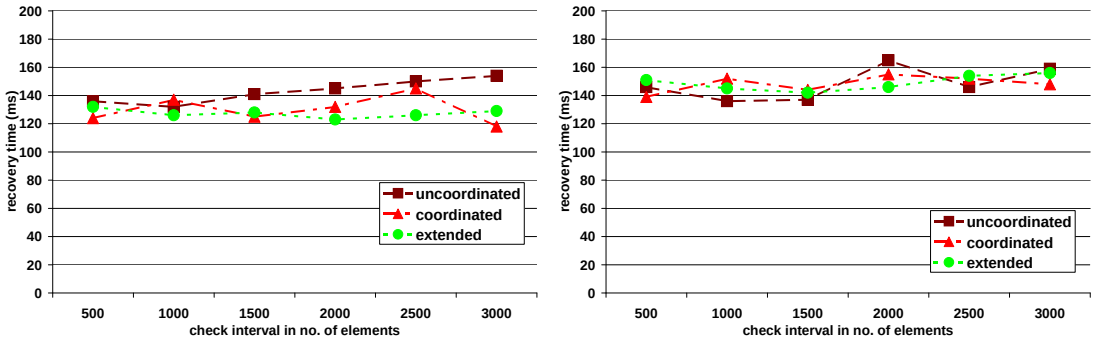


(d) Stream Process 4

Figure 7.21: Recovery Time Single Failure

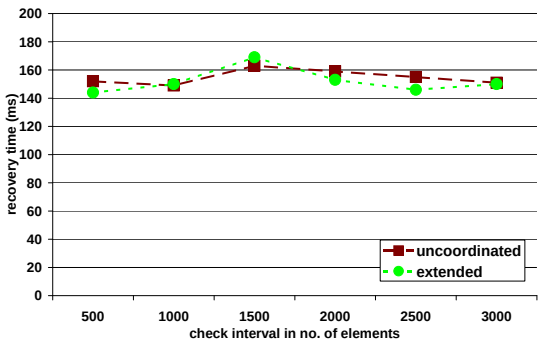
checkpoints is increased. Given this fact, longer intervals between checkpoints cause longer catchup times.

Also in CPU utilization (see Fig. 7.23 and Fig. 7.23) there is no significant variation between the settings. In JVM memory consumption (see Fig. 7.25 and Fig. 7.26), we see a slightly lower memory consumption for the coordinated and extended settings compared to the uncoordinated setting in most experiments. This conforms with the fact that also during failure-free runtime the JVM memory consumption is significantly less for Stream Process 1 and Stream Process 2.

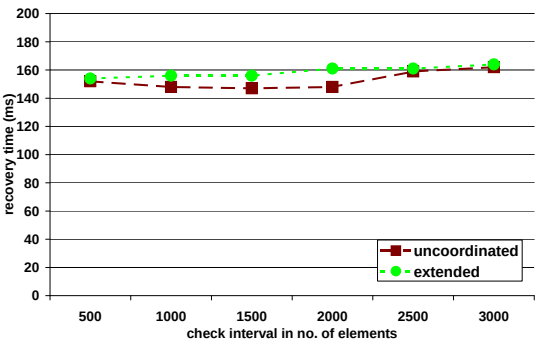


(a) Stream Process 1

(b) Stream Process 2

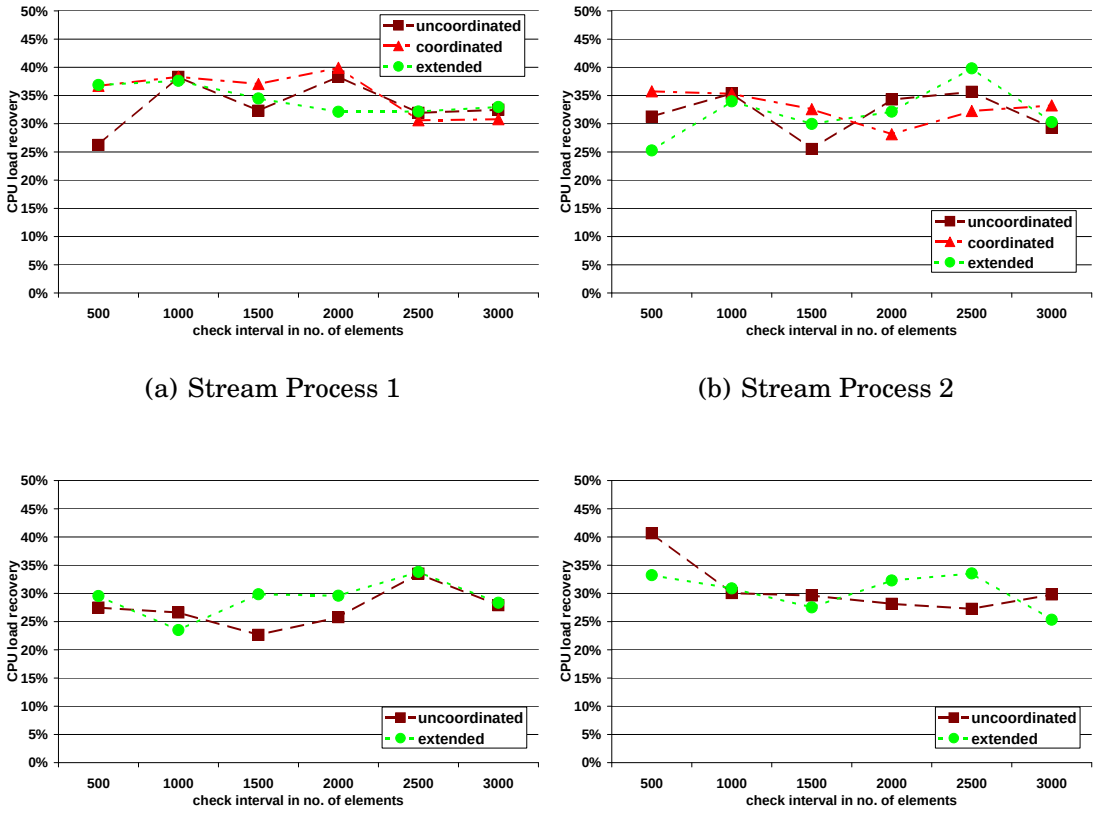


(c) Stream Process 3



(d) Stream Process 4

Figure 7.22: Recovery Time Multiple Failure



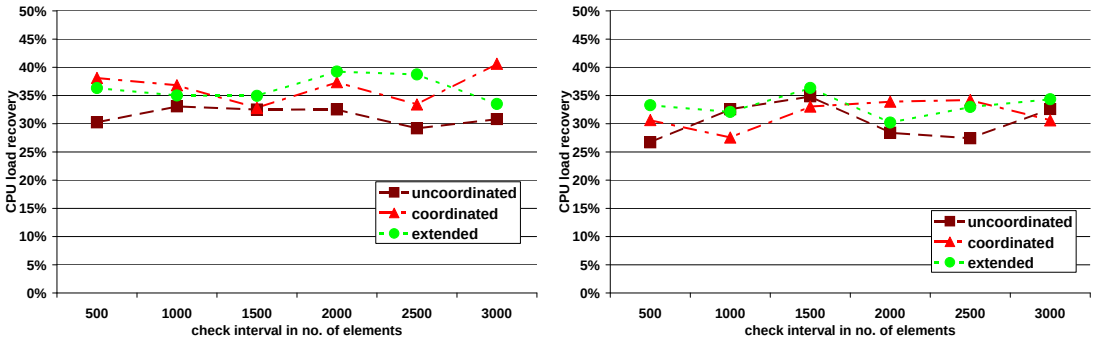
(a) Stream Process 1

(b) Stream Process 2

(c) Stream Process 3

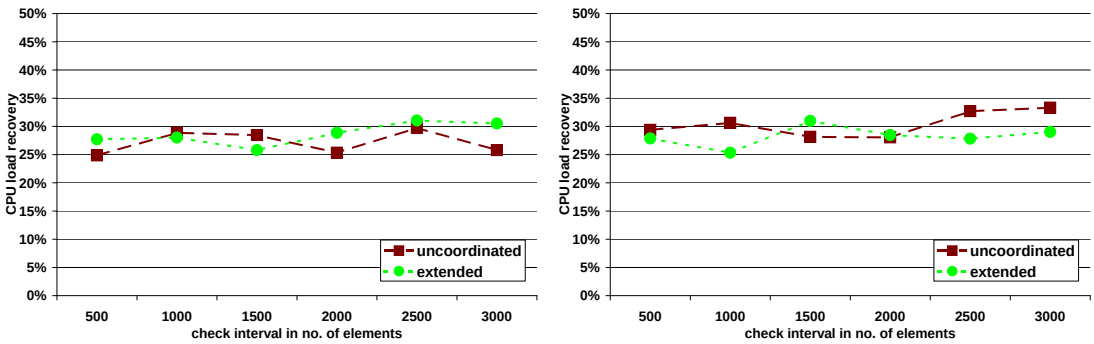
(d) Stream Process 4

Figure 7.23: CPU load during Recovery Time Single Failure



(a) Stream Process 1

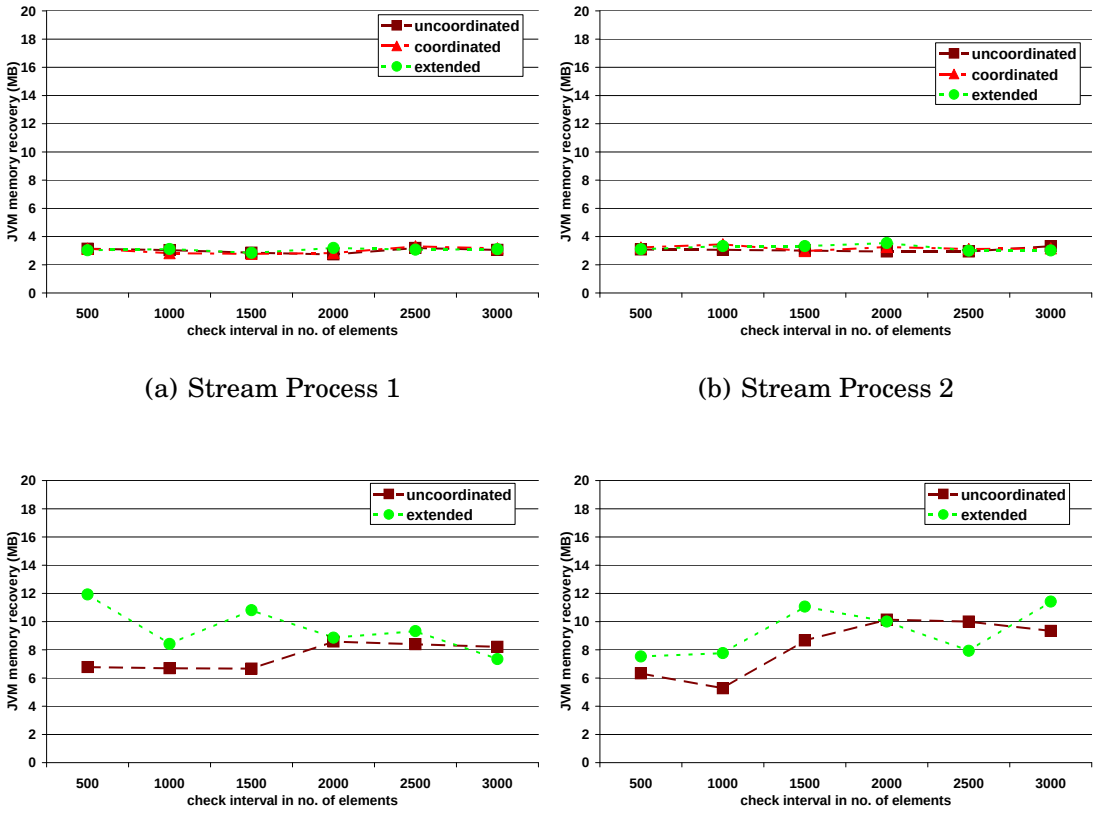
(b) Stream Process 2



(c) Stream Process 3

(d) Stream Process 4

Figure 7.24: CPU load during Recovery Time Multiple Failure



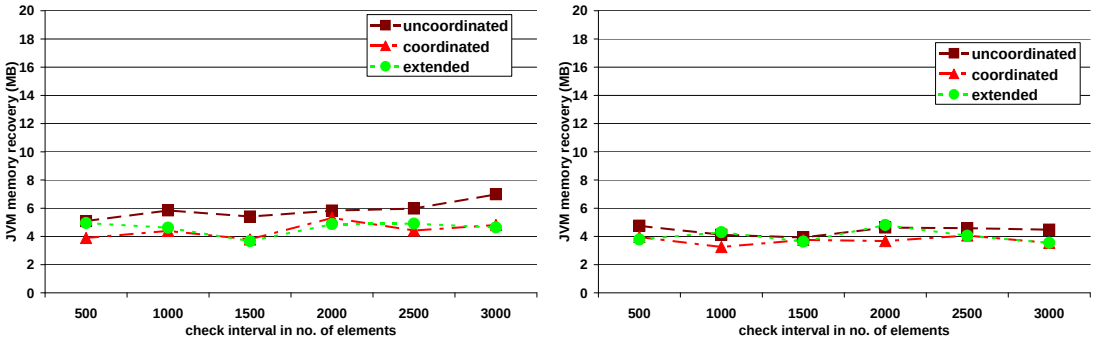
(a) Stream Process 1

(b) Stream Process 2

(c) Stream Process 3

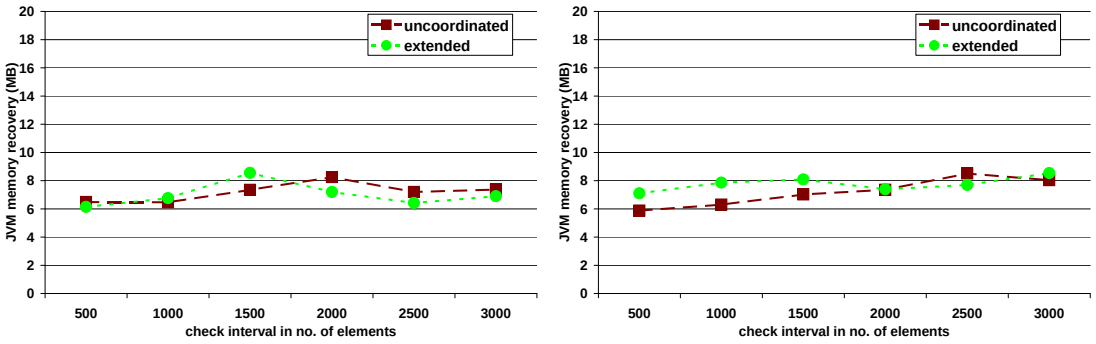
(d) Stream Process 4

Figure 7.25: JVM memory consumption during Recovery Time Single Failure



(a) Stream Process 1

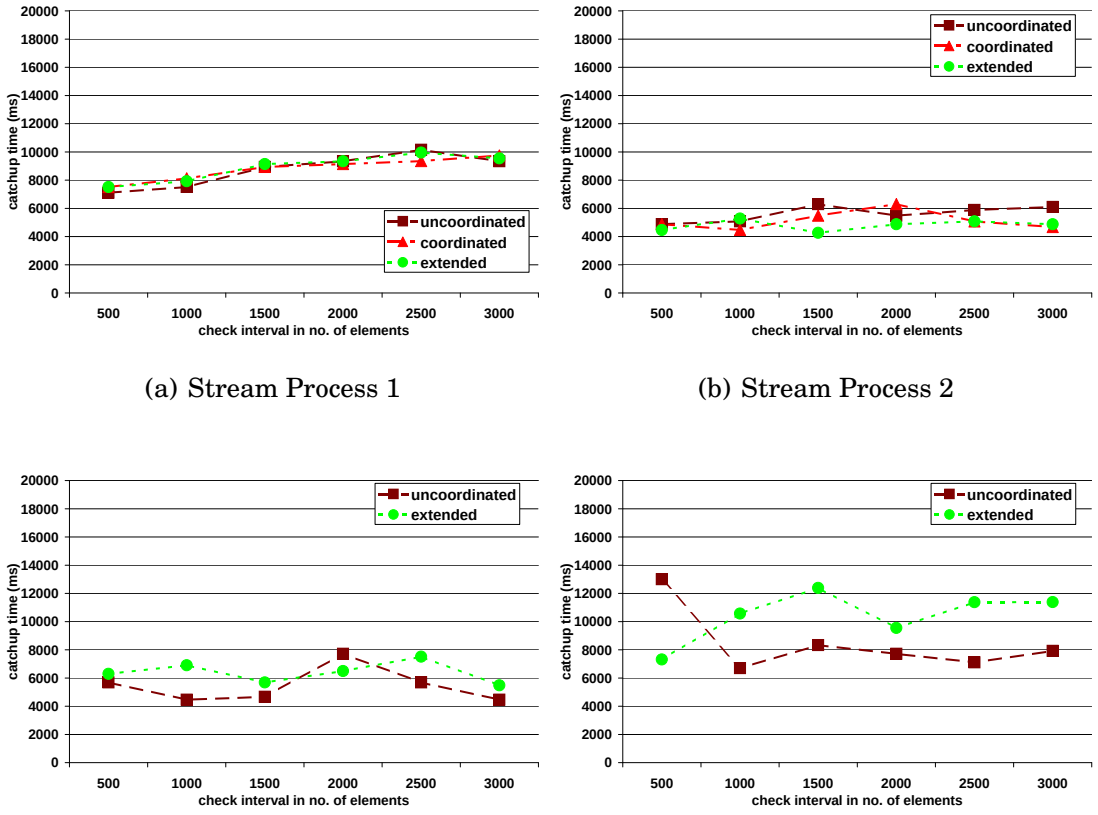
(b) Stream Process 2



(c) Stream Process 3

(d) Stream Process 4

Figure 7.26: JVM memory consumption during Recovery Time Multiple Failure



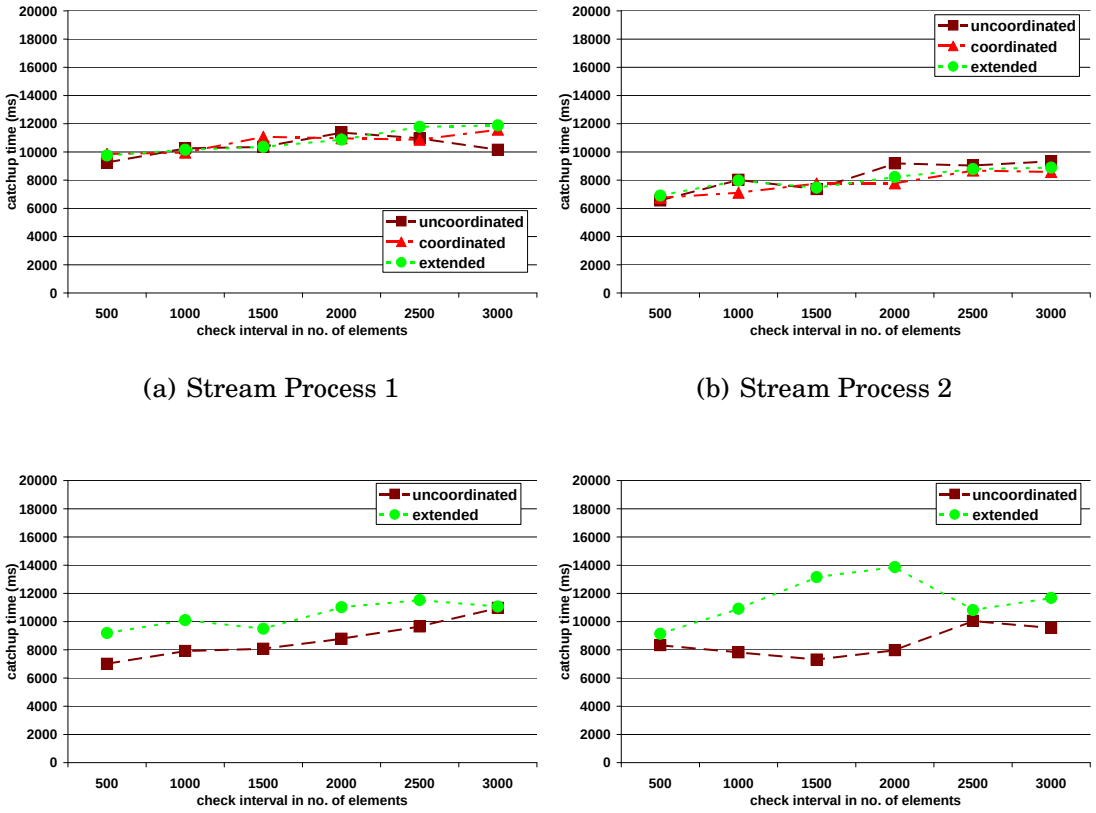
(a) Stream Process 1

(b) Stream Process 2

(c) Stream Process 3

(d) Stream Process 4

Figure 7.27: Catchup Time Single Failure



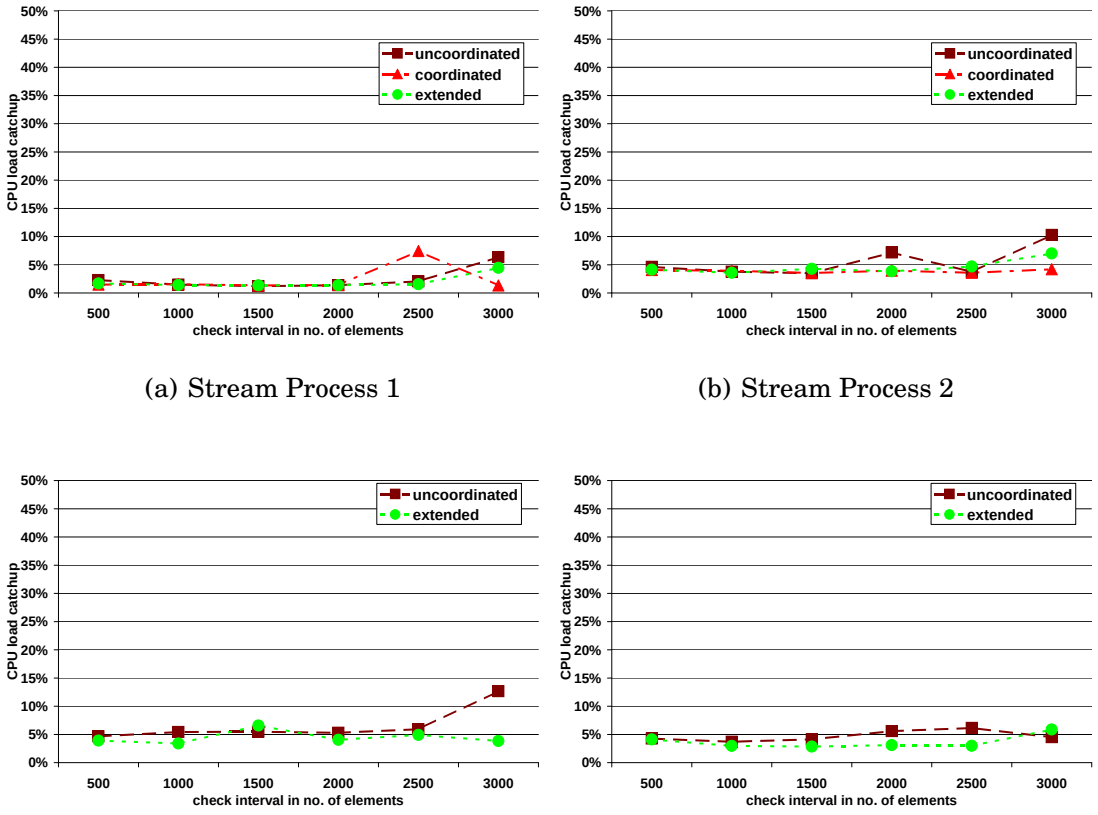
(a) Stream Process 1

(b) Stream Process 2

(c) Stream Process 3

(d) Stream Process 4

Figure 7.28: Catchup Time Multi Failure



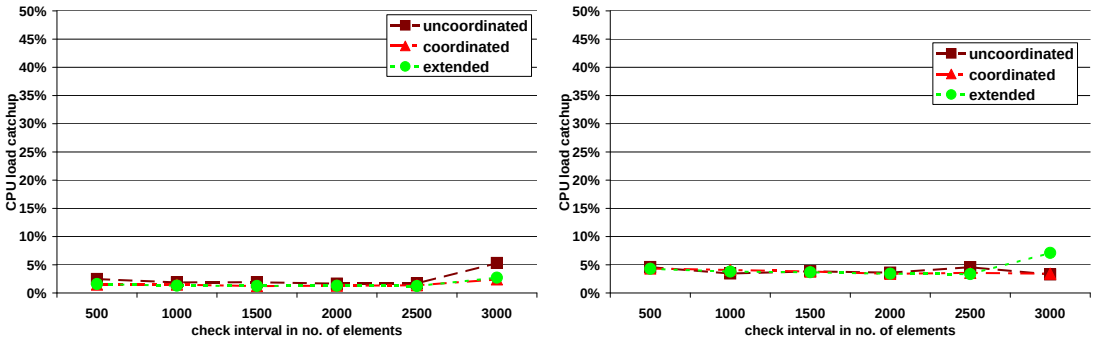
(a) Stream Process 1

(b) Stream Process 2

(c) Stream Process 3

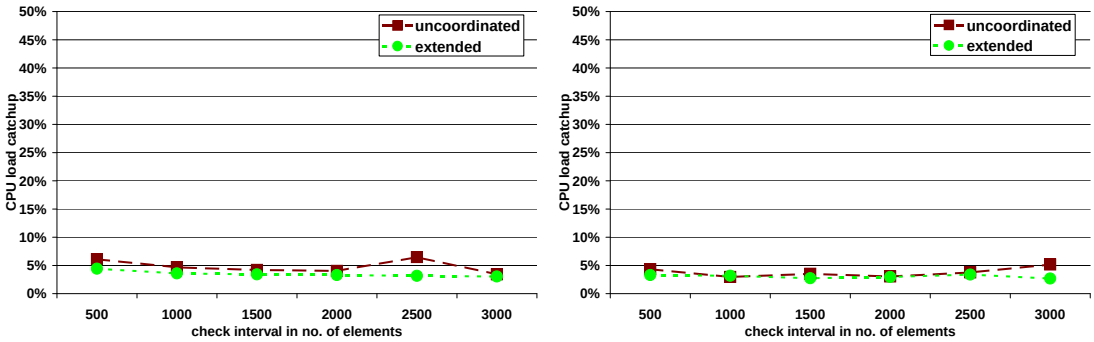
(d) Stream Process 4

Figure 7.29: CPU load during Catchup Time Single Failure



(a) Stream Process 1

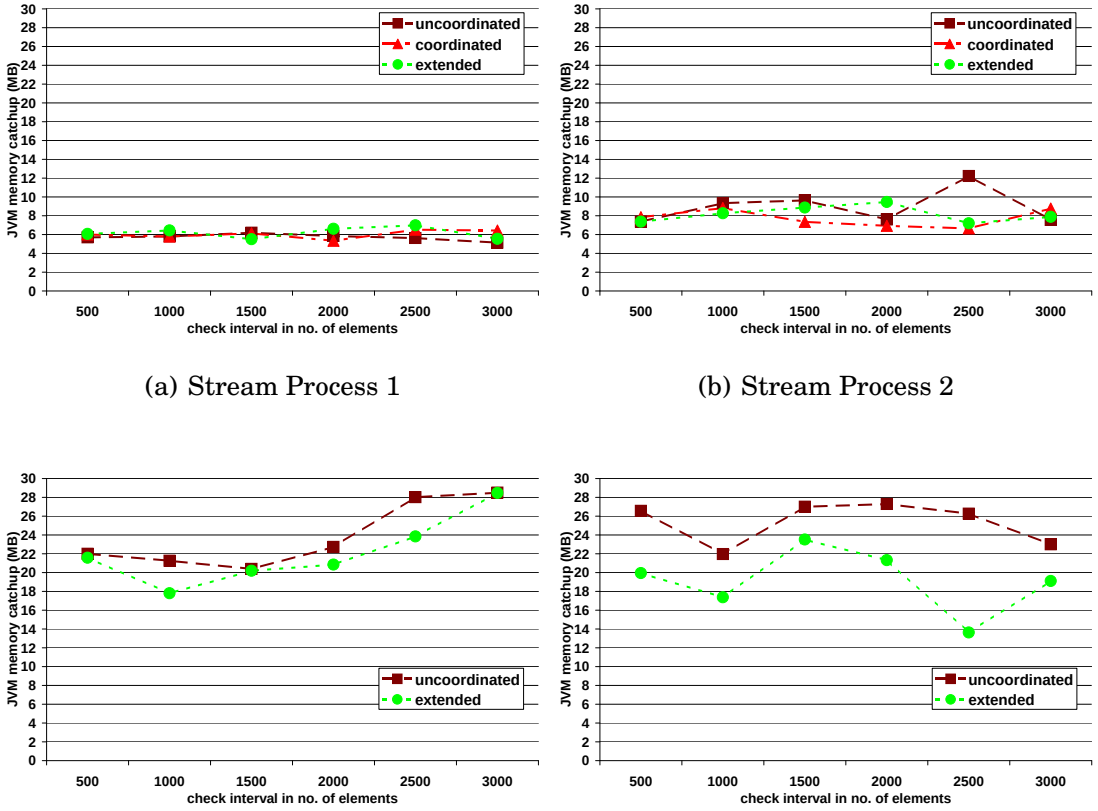
(b) Stream Process 2



(c) Stream Process 3

(d) Stream Process 4

Figure 7.30: CPU load during Catchup Time Multi Failure



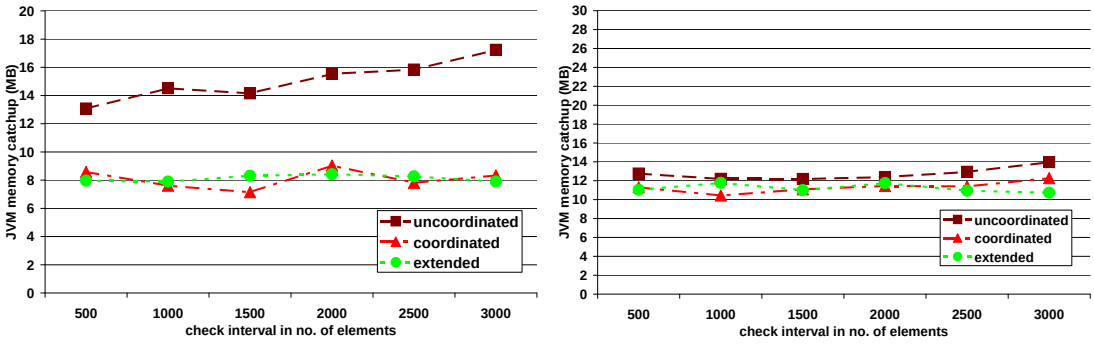
(a) Stream Process 1

(b) Stream Process 2

(c) Stream Process 3

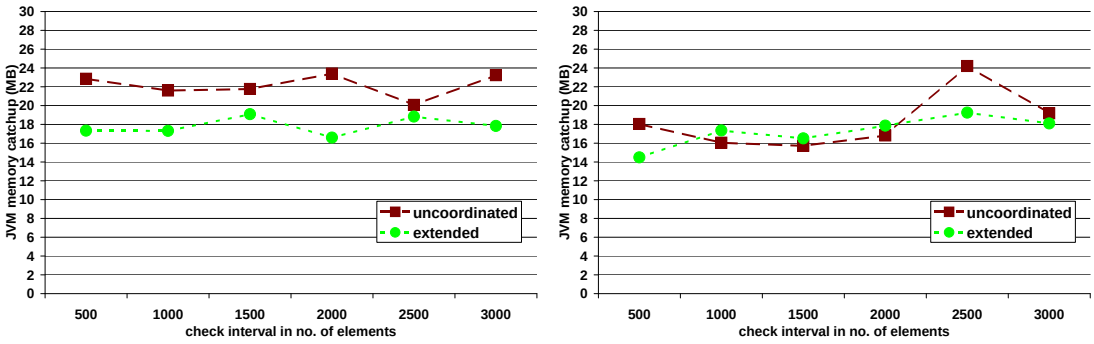
(d) Stream Process 4

Figure 7.31: JVM memory consumption during Catchup Time Single Failure



(a) Stream Process 1

(b) Stream Process 2



(c) Stream Process 3

(d) Stream Process 4

Figure 7.32: JVM memory consumption during Catchup Time Multi Failure

7.2.7 Summary

The evaluations in the mobile and stationary environment have shown that the ECOC approach and extended ECOC approach performs significantly better than the uncoordinated setting, which uses the standard passive standby approach. In particular, ECOC dramatically reduces the network overhead, which is the major drawback of the uncoordinated passive standby approach. Additional measurements demonstrate that ECOC does not result in higher memory consumption than the uncoordinated checkpointing approach. For the handling of single and multiple failures, the experiments have shown that ECOC and extended ECOC are about equal in performance compared to the uncoordinated approach in recovery and catchup phase.

Moreover, the experiments have shown that OSIRIS-SE implements reliability of DSM processing for various platforms in an resource efficient way. The implementation fulfils the limited-delay, lossless, and intra-stream order preserving reliability levels of Chapter 5. Overheads due to reliability algorithms for data stream processing are affordable even at mobile devices. Due to the ECOC and extended ECOC approach network overhead has been minimized. Furthermore, the experiments have shown that memory overhead is the second major overhead due to reliability algorithms. Nevertheless, increasing memory availability even for mobile devices is relieving this issue. CPU overhead is still measurable in the mobile environment but evanescent in the stationary environment. A reason for this, CPU utilization is caused by network access of mobile devices.

8

Related Work

In this chapter, we present related work in various areas. We start with early work on reliability of distributed systems. In this section, we investigate how to achieve reliability in the domains of distributed databases, distributed workflow management, Peer-to-Peer computing, computer networks, sensor networks and middleware.

Furthermore, we describe related research in the area of DSM and in particular how this research handles reliability issues.

Moreover, this chapter presents research in the e-health field. In this section, we present application oriented research and derive motivation for DSM and reliability demanded by such applications. Finally, we give an overview of already existing product solutions in the area of e-Health applications with relevance to DSM.

8.1 Reliability of Distributed Systems

The reliability approach presented in this thesis is based on process-pairs [Bar81], which describes a model of primary and backup processes. The primary process checkpoints all requests to the backup process, so the backup has all information necessary to take over control in the case the primary fails. In addition, early database research work [Gra78] has introduced the concepts of checkpoints in transaction processing to allow for faster recovery. This approach has been widely adopted in distributed database research, for example in the Tandem architecture [BGH86]. Furthermore, work of Chandy & Lamport [CL85] emphasized on performing checkpoints in a distributed system in order to get a meaningful global state. Based on this, work of Baldoni et al. [BQF99] emphasized on algorithms to take consistent check-

points for general distributed systems. Elonazhy et al. [EAWJ02] have presented a survey of work on rollback recovery protocols in message-passing systems. Distributed DSM systems are a special kind of message-passing systems, therefore many of these protocols can be applied to DSM as well. Nevertheless, based on the presented DSM model, we have tailored these general approaches. For example the domino effect does not appear in our approach. Firstly, because our checkpointing approach only keeps the most recent checkpoint in order to reduce the load on the backup host. Therefore, rollback propagation a prerequisite for the domino effect is not possible. Still our proposed algorithm guarantees that recent checkpoints are consistent and support lossless reliability. This is achieved, by relaxing the notion of reliability with respect to our deterministic system model. Concurrency between stream processes can only become an issue when taking the concurrent side effects on external systems into account which we have not done within this thesis.

Research on distributed databases [GMK88, GMP90] has handled the conflicting goals of correctness and availability by applying sophisticated backup algorithms. Due to the "single producer/multiple consumer" nature, concurrency control is not stressing availability of DSM. Different stream processes can be processed individually without taking the execution of other stream processes into account.

Parallel database systems [DG92] spread data processing tasks across a cluster of databases for scalability and reliability reasons. Matching this approach to the DSM scenario, a stream process as a set of logical linked operators explicitly describes pipelined parallelism, which is leveraged in our proposed DSM infrastructure. Exploiting partitioned parallelism as described in [DG92] narrows the operator model, to operators that allow for independent processing of a chunk of elements, whereas the membership to a chunk is defined by the element content or timestamp information. This restriction cannot be applied to operators in various application domains, e.g., in telemonitoring for healthcare.

Workflow management research also addresses reliability and availability. Exotica [KAGM96] and Opera [HA99] investigated different process instance backup strategies based on the process-pairs approach. As these projects were depending on central workflow control and central process databases, further work within the OSIRIS [SWSS03, SWSS04] project moved towards decentralized control based on replicated meta-information. This approach leveraged the distribution of process execution for reliability and availability, due to redundancy of services and providers. Work on transactional processes [SABS02] handled reliability and correctness on process specification level. Current commercial workflow management products, like IBM's MQ

Workflow [IBM04] still depend on central workflow control and hence on a central database, but high availability is achieved by having a redundant reliable message queue and a central database running on a high availability cluster.

Peer-to-Peer and Grid computing research has also applied the process-pairs approach. The distributed batch processing system Condor [LTBL97] provides checkpointing for fault tolerance by allowing UNIX processes to migrate to other nodes. P2P research [SFG+02, NW03, HK03] applies redundancy for fault-tolerance to routing and lookup. These works are potentially beneficial for future research in order to improve reliability for the repositories and on the transport level of OSIRIS-SE.

The *P-Grid* [A+03] is a Peer-to-Peer project and covers interesting issues that are also relevant to our infrastructure research. P-Grid provides an advanced, fully decentralized Peer-to-Peer infrastructure. A Peer-to-Peer lookup system offers access to global available information replicated among the available peers. Reliability is achieved by redundant replications. Additionally, P-Grid also covers the problem of updates on this distributed replicas.

Reliability of computer networks is generally achieved by rerouting traffic to alternative path as in OSPF [Moy98] and BGP [Rek95]. At transport layer reliability is provided by retransmission and congestion control as in TCP [Pos81]. Furthermore, research [FJL+97, LP96] is applying these concepts also on reliable multicast communication. Multicasting provides an efficient way to disseminate data to a group of receivers. Receiver driven acknowledgment assures lossless delivery of messages. A drawback of these approaches is the additional control overhead and the strict reliance on the round trip time which is not suitable for wireless connections. In point-to-point transport, the *resilient overlay networks* (RON) [ABKM01] allow Internet applications to detect and recover path outages. RON offers improved fault-tolerance compared to wide-area routing protocols like BGP because the RON nodes are able to exploit physical path redundancy by self-organization based on path monitoring. Recent research in reliability of networks focus on self-healing protocols [YSM07] by using artificial chemistry paradigms. This research focuses on software reliability and assumes the existence of a high rate of soft-errors. This means that the targeted hardware environment produce sometimes faulty results. In order to compensate these faults an artificial chemistry approach applies redundant paths in protocol software. Contrarily to our thesis, this approach is non-deterministic. At the current state, lossless transport of data messages cannot be guaranteed and is also not the intention of this approach.

Work in the area of reliability in wireless sensor networks focuses on transport layer reliability [SH03, WCK02] with support for limited-devices. Trans-

mission errors are handled by retransmission between neighbors or discovery of best routes within the sensor network. Furthermore, declarative failure recovery [RG07] research in sensor networks allows the programmer of sensor nodes to specify checkpoint and rollback behavior as annotations within the code. Contrarily in our research, we assume operators offer an interface which allows the infrastructure to transparently access their internal state for checkpointing. Research in wireless sensor networks offer various valuable building blocks required for reliable DSM applications as considered in this thesis. But this is not sufficient, reliability is also needed at the higher abstraction level of data stream operators.

Approaches in the area of reliable middleware [KIW05, NGYS00, DMM+02] propose central coordinators in a process-pairs approach, discrete, non-streaming function calls, and reliable messaging. Services are encapsulated by a reliability layer in order allow for transparent fail-over based on logging/checkpointing techniques. The applicability of this research more suitable for server clusters with Ethernet connections rather than mobile environments with wireless connections.

8.2 Data Stream Management Research

In what follows, we present a historical evolution of DSM research and introduce major DSM projects in more detail. In particular projects that provide beneficial features for telemonitoring applications in healthcare.

Early work in the field of DSM derived from sequence database research [SLR96]. The authors suggested a declarative sequence query language and presented a combined nested design in order to combine the processing of sequence and relational data in a single system. The combined system was still processing previously stored data implemented as large arrays on disks.

The *Tribeca* [SH98] project was driven by the demand to process large amount of traffic data coming from telephone and computer networks. The novel idea was to operate directly on the live-data and not on stored data due to large volumes of data.

Other DSM research evolved from event-driven query systems [RW97]. The OpenCQ [LPT99] extended event-driven systems in order to deal with continual queries, that monitor user's interest and whenever certain thresholds are reached the information is delivered to the user. Data stream processing in this project is limited to simple filtering tasks. Typical applications are notification services that inform about newly available information of interest in the world wide web.

The *NiagaraCQ* [CDTW00] project emphasized on the scalability issue of DSM for internet information notification applications. Their assumption is that large number of users are posing continuous queries. Moreover, the different continuous queries have common sub-expressions. Based on these assumptions, the DSM system is able to create groups of queries with common sub-expression whereas the common parts are executed only once in order to improve scalability. Obviously, for the healthcare scenario and other DSM applications these assumptions are not valid. Each stream process is processing very specific data streams coming form the associated patient. In this case, there are no common sub-expressions that are subject for optimization.

The *COUGAR* [YG03] project is aimed to support DSM for sensor networks. The project describes a platform for continuous query processing over ad-hoc sensor networks. Nodes in sensor networks are in general small embedded devices with sensors and additional computing and wireless communication facilities, also called *motes*. The intended application scenario, e.g., environmental monitoring, consists of thousands of motes connected wirelessly with each other. The project was motivated by the idea of applying declarative queries to data processing in sensor networks. In order to reduce energy consumption processing imposed by wireless communication, filtering of sensor data was moved as close as possible to the sensor mote. This optimization, called in-network aggregation, reduced energy consumption for transfer of sensory data because of less energy consumption needed for computing. Moreover, multi-query optimization as in NiagaraCQ was applied in order to share processing results of similar queries. Continuous queries and their optimizations in this project are limited to a few basic operations. These operations are not sufficient to support stream processes needed for various application domains, e.g., healthcare applications. Moreover, the important issue of reliability within the fragile sensor network has not been discussed in this project.

The *STREAM* [BBD+02, BW01] project describes a DSM system for high data rates and large numbers of in parallel executed continuous queries. Users are able to register and deregister queries in the system by using a declarative query language, called *Continuous Query Language (CQL)* [ABW03]. Internally, CQL is compiled into a query plan based on continuously running operators. Research is focused on the design of the query language. Based on the operations defined in the query language, optimization for query processing are presented within the project. These optimizations include operator scheduling, reordering of operators within the execution plan, merging of common parts between different queries, and load shedding in order to meet delay requirements in cases of high system load. The project has not considered distributed DSM processing so far and rather sticks

to a central implementation of the DSM system. The operations presented in the CQL language and used as basis for optimization research are basic operations, like filters, joins, and aggregation functions on data streams. For the various applications considered in this thesis, these can form set of basic operations but generally they are far to limited to cope with the different processing demands. In this thesis, we support general application specific operators and also DSM execution needs to be performed in a distributed setting.

The *Aurora* [C+02] project allows for user defined query processing by placing and connecting operators in a query plan. As for other projects Aurora is focusing on a limited set of well-defined DSM operators. As proposed by our approach, Aurora offers a graphical user interface to design continuous query graphs in an boxes and arrows approach. Interaction with relational databases is achieved by applying storage points, called *connection points*, within the query graph. Historical data derived from these connection points can be used for other continuous queries again. Moreover, Aurora allows for deficiencies of DSM processing. The required result quality is defined by *quality of service (QoS)* definitions for latency, value accuracy, and loss tolerance. Query optimization algorithms are trying to improve the overall QoS of the system. Important research aspects emphasized within the Aurora project are efficient operator scheduling and load shedding in order to keep the required QoS in cases of high system load. Conceptually, Aurora started as single node architecture and extensions like *Aurora** and *Medusa* [CBB+03] also emphasized on DSM in distributed environments. Aurora* is a connection of multiple Aurora nodes to one query network. If a node has high load, it can offload operators to other nodes to reduce its load and improve the overall QoS. The follow-up project *Borealis* [AAB+05] is presenting a distributed DSM infrastructure based on Aurora* and Medusa. Borealis offers three new main features: dynamic correction of query results, dynamic modification of continuous queries based on current results, and highly scalable optimization for heavy load situations. Also reliability aspects of DSM are discussed within the Aurora and Borealis projects [HBR+05, BBMS05, HXCZ07] which are presented in detail in the following Section 8.2.1 of this chapter.

Other important projects in the area of DSM and sensor data processing have been conducted at the U.C. Berkeley. The *Smart Dust* [WLLP01] project started with the vision of having sensor networks consisting of millions of tiny nodes with limited hardware functionality at the size of dust particles communicating with each other and thereby processing sensor readings. Closely related to this project, the *TinyOS* [HSW+00b] project emphasized on developing a tiny, event-driven, modular operating system for the tiny sensor devices. The goal is to free the programmer of sensor processing applications from

hardware specific issues and to support systematic advance of sensor processing applications. *TelegraphCQ* [C+03] is a DSM project with special focus on adaptive query processing. Since the projects at U.C. Berkeley are based on the *Smart Dust* vision, all consideration were dealing with distributed DSM processing from the very beginning. The subproject *Fjords* [MF02] offers an infrastructure for inter-module communication between an extensible set of operators enabling static and streaming data sources distributed across a loosely-coupled network of nodes, i.e., a sensor network consisting of tiny motes. The infrastructure can manage multiple queries over many sensors and has limited resource demand. Performance evaluations are based on real-world data coming from a road traffic monitoring application. Another subproject called *Eddies* [AH00] describe an adaptive query processing operator usable within the Fjords infrastructure. Eddies is continuously processing input data streams and is routing the data stream elements through the operator graph. One important feature of Eddies is the dynamic reordering of operators on the fly based on current workload, input rates and current selectivity of operators within the DSM infrastructure. Eddies is analyzing the current execution state of the query and if possible reordering of operators is applied. In the extreme case this can be done for each incoming data stream element. Obviously, the work of Eddies is depending on the very specific features of each operator in the query graph. Contrarily to our work, this work is tailored for a certain set of available basic data stream operators, i.e., different kinds of filters and joins. This work is not applicable to a DSM infrastructure which supports the execution general data stream operators, e.g., operators used for body signal analysis in healthcare. In addition to Eddies, the *Flux* [SHCF03] emphasize on load balancing and reliability within data stream processing in a distributed environment. Conceptually, Flux is an operator as Eddies which is executed within the Fjords infrastructure. In order to offer load balancing and reliability the Flux operator is placed between a producing and a consuming pair of operators in the query graph. More details on the reliability aspects within the TelegraphCQ project [SHCF03, SHB04] are presented in detail in the following Section 8.2.1 of this chapter.

The D-CAPE [LZJ+05] project is also emphasizing on distributed DSM processing. Unlike other projects in this field, D-CAPE is focusing on cluster environments connected by reliable high speed networks. Research issues within the project are load shedding, operator scheduling, and migration of stateful operators [ZRH04]. Unlike as proposed in this thesis, operator migration is only used for load balancing and not used to achieve reliability. Moreover due the cluster environment with high network bandwidth, efficiency of operator migration in terms of communication overhead is not discussed in this work.

The *PeerCQ* [GL03] system offers a decentralized peer-to-peer DSM infrastructure. The intended application scenario is internet scale information monitoring. For this reason, the continuous query itself is rather simple compared to a stream process in an healthcare application. The purpose of a continuous query focused on information filtering. For this reason, there is no need to split execution of a single continuous query across multiple peers. The work is focused on capability-sensitive assignment of monolithic continuous queries to appropriate peers. A certain degree of robustness is given due to the large amount of available peers but reliability issues regarding the execution of a single query are not discussed within this work.

DFuse [KWA+03] is a framework for distributed data fusion. DFuse also allows for distributed query processing by connecting operators. The intended application scenario is hierarchical sensor data processing in an environment containing embedded and stationary device nodes. The framework offers a data fusion API, which supports the development of data fusion applications. DFuse puts main emphasis on distributed power-aware placement of operators across nodes in wireless ad hoc sensor network, where routing has also to be considered in order to minimize transmission costs.

The *StreamGlobe* [SKK04] project aims at providing distributed DSM in heterogeneous peer-to-peer network environments containing mobile and stationary nodes. Data streams in StreamGlobe are expressed by using XML and queries by using XQuery. The infrastructure of StreamGlobe is based on the Grid reference implementation *Globus Toolkit* [The03]. Moreover, StreamGlobe puts an additional Peer-to-Peer network layer on top of the Grid infrastructure that is basis for DSM functionality. Meta data distribution is based on *ObjectGlobe* [B+01]. Continuous queries are posed as subscriptions of users. By using mobile code in subscriptions, users are able to pose complex queries with user-defined operator code. Optimizations are based on data stream filtering where unneeded parts of the data stream are removed and data stream clustering where common result streams of different queries are joined in order to reduce execution overhead.

Pipes [KS04] offers a variety of basic building blocks for DSM within a Java library. The library approach of Pipes enables the creation of a tailored DSM system for a specific application scenario. Pipes integrates a query construction framework and covers the functionality of the CQL [ABW03] query language. Additional frameworks for scheduling, memory management, and query optimization are also provided. Research issues within the Pipes project are data stream mining, e.g., runtime analysis of data stream characteristics by using Wavelets. Other work is focusing on building up semantic operator algebra of a fixed set of basic operators for DSM with special regard to temporal issues. Recently, research in this project also addresses dy-

namic query plan migration [KYC+06] in order to deal with load fluctuations within the network.

The recent project *Xtream* [DTAK07] at ETH Zurich is also focusing on a general purpose data stream processing infrastructure. The infrastructure defines an operator interface and therefore allows for application specific DSM processing as also pursued in the work of this thesis. Whereas, other projects in the field focus only on a specific set of basic DSM operators. The intended application scenario considers a smart home which integrates technologies for security, life-style, and communication. Moreover, the infrastructure is distributed, supports mobile and stationary devices, and allows for dynamic deployment of services. Closely related to this project is *SwissQM* [MRD+07]. SwissQM focuses on sensor network applications and emphasizes the problem of integrating embedded devices in the DSM infrastructure. For this reason, the project proposes a virtual machine for embedded sensor devices, which is able to execute a platform independent bytecode language with platform independent data types. Whereas, TinyOS [HSW+00b] is using an device dependent C-dialect with hardware dependent data types. Moreover, the virtual machine supports event-processing at sensor nodes, finite state machine DSM processing operations, and user defined functions. This offers a powerful platform for higher level applications and allows to integrate sensor networks into the Xtream infrastructure. Reliability aspects of DSM are currently not considered in both projects. In the approach of this thesis, we are focusing on Java-enabled devices and therefore we are using the Java virtual machine to achieve platform independence and to ease integration of mobile devices.

Another project with regard to continuous data stream processing is called *Information Filters* [FK05]. Information filtering is dealing with the problem of filtering and routing information to their recipients in an environment where information is continuously produced. Compared to traditional DSM research, the task of processing of the information is not emphasized rather than filtering of messages by their content. In particular, this work focuses on the scalability of information filters. Users pose their information interest in *profiles*. These profiles are applied to incoming messages to check if a message is for interest of the user. The work improves scalability of filtering in terms of number of profiles and incoming messages. Messages within this work are usually considered as XML messages but the approach is not limited to XML. Improved scalability is achieved by applying batched processing. Traditional processing of information filter queries for each incoming message indexes for predicates in order to get matching profiles whereas batched processing applies indexed processing for batched groups of incoming messages.

8.2.1 Reliability Aspects in DSM Research

Reliability of DSM systems is still underrepresented in research, besides the work done in this thesis only two groups are focusing on particular aspects of reliability in DSM. In what follows, we discuss the particular differences of the approaches.

Within the *Aurora* [C+02] project high availability has become of research interest. Various algorithms and a model of reliability in DSM have been discussed in [HBR+05]. The work introduces different *recovery styles*. *Precise recovery* is considered as having identical output compared to a system without failure considering a fragment of the query plan. *Rollback recovery* considers the case where duplicates are produced in case of failures and *gap recovery* considers gaps in output data streams of a system suffered from a failure. Moreover, the work gives a *classification* of DSM operators. Firstly, *arbitrary operators* have no processing constraints. Secondly, *deterministic operators* are producing the same output streams, when starting from the same internal state and receiving the same input streams. Thirdly, *convergent-capable* operators are producing the same output streams, when starting from an empty state and receive the same input streams but the input streams are starting earlier in time as the start of the operator instance. Fourthly, *repeatable operators* have no internal operator state which means that they are producing the same output streams when feed with the same input streams without regard to previously seen parts of input streams.

Compared to our DSM model, we put more emphasis on a detailed formal model of reliability within DSM processing and presented a more fine grained model based on *delay, loss,* and *order* of data stream elements within the recovered system. Our model offers applicability to various application domains where high reliability is necessary. For this reason, we consider convergent capable operators and the possibility to converge back to correct data stream results when recovering operators from empty state only as an optional feature. The reason for this is twofold. Firstly, there would be a gap in the result stream which is filled with incorrect results until convergency is reached. Secondly, only for specific DSM operations these assumptions are valid. Since our model is designed to work with general operators and not limited to a basic set of well-known operators these classification is not applied. In this thesis we consider operators to be deterministic as the *only* restriction to operator behavior. Still, work in in [HBR+05] states three reason for non-deterministic operators. The first reason is *time dependence* of processing. The DSM model proposed in this thesis considers two kinds of time explicitly. One kind is the processing time or global timestamp τ. In a real-world application only at the sensor node this timestamp is meaningful to the application because there it

corresponds to the actual time of measurement, e.g., the time when patient Fred had a heartbeat of 73 beats per minute. At all subsequent operator stages within the data-flow of the stream process, this timestamp is subject to additional delays and therefore dependent on the current load and failure situation within the DSM infrastructure. For the application, the infrastructure is considered to be transparent therefore the processing delays are *not* allowed to have impact on results of DSM processing. Obviously, the delays are still important for the application but only in the sense that the maximum tolerable delays are not exceeded. Given this fact, means that all necessary time information needed for application specific DSM processing is coming from the sensors and is considered as part of the payload information. The second reason for non-determinism is *arrival order* of data stream elements and is related with the second kind of time in our DSM model. Arrival order corresponds to the stream time or sequence number of a data stream element. Due to the fact that the underlying infrastructure delivers data stream elements in correct order also this reason for non-determinism is eliminated. Lastly, only randomization in DSM processing is left as a reason for non-determinism. We state that with regard to our intended application scenarios randomization is not an issue because reproducible results are important.

In addition, work in [HBR+05] is discussing different recovery approaches. The recovery approaches emphasize on single-node fail-stop failures and consider that a failure affects a complete node and *all* DSM processing done at the affected node is moved to a single backup node. The work presents *passive standby* as periodically sending changes of the current state to the backup node so that the backup is able to take over in case of a failure. This approach is not discussed in detail in this work and rather described for comparison. Secondly, *upstream backups* are discussed as an improvement of the passive standby approach and is presented in more detail. The upstream backup approach keeps *all* necessary backup information at the *upstream node* and *no* periodically state information is sent to the *backup node*. Upstream nodes are nodes that produce input streams for the affected node. In this approach, the output queues of upstream nodes are keeping all necessary data stream elements so that the backup node can recover DSM processing from *empty* state. Consequently this approach needs the DSM infrastructure to have a mapping between input and output data stream elements for each participating node. Moreover, the consequences of this approach for multiple failure scenarios are not discussed. An important constrained imposed by this approach is that operators are convergent capable which is in general not applicable. Finally, for reasons of completeness and for comparison, the *active standby* approach is presented. In this case, the backup node receives continuously all input data streams of the primary node in parallel. The DSM processing is performed

in parallel at both nodes. For this reason, the backup node has an output queue which has to be trimmed as for the primary node. In case of a failure the backup node can immediately replace the failed node. The detailed comparisons between the three different approaches come out with the following result. Active standby has a high network and resource overhead but offers very fast recovery in case of failures. Passive standby is considered as not well suited for DSM processing because of similar network overhead than active recovery with worse recovery performance. As an outcome of this work, upstream backups are proposed as best suited reliability approach because of lowest overhead in the single failure case. However, the work admits that for precise recovery of arbitrary operators, as needed in e.g., patient monitoring, passive standby is the appropriate approach.

Comparing these reliability algorithms with the work presented in this thesis, we state that uncoordinated checkpointing is comparable to the passive standby approach. Still, there are important differences in our work. Firstly, we assume general deterministic operators without any further constraints on their behavior. Secondly, we assume multiple failures as normal and likely event with regard to the considered pervasive computing environment. Thirdly, we apply the checkpointing approach at the level of DSM operators and not at the level of nodes. This means that every operator instance running at a node may have a different backup node. Obviously in case of a failure, this allows for a better utilization of the remaining DSM network since the more fine grained load redistribution. Moreover, in this thesis we present efficient and major extensions of uncoordinated checkpointing in order to reduce the overhead of the reliability algorithm and improve the recovery time.

Further work [BBMS05] on reliability of DSM is presented in the context of Borealis [AAB+05] the successor project of Aurora. The work proposes an active standby approach where replicated nodes of the DSM are processing the same input streams and producing output streams. An important feature of this approach is to allow for reduced result quality. This means that due to a failure a node has missing input data streams. In this case, the node is still allowed to process the incoming data received at other input streams and produce *tentative* results. Tentative results are subject to later correction when the correct results are available. Again, tentative results are most appropriate for convergent capable operators which may lead automatically to correct results after some time. An advantage of the approach is to allows for user-defined tradeoff between availability (having results very fast but may be not correct) and consistency (having correct results but may be delayed).

Comparing this approach with the approach of this thesis, we see that an active standby approach is applied in opposition to checkpointing. Moreover,

for our intended application scenarios where lossless reliability is demanded and short delays are tolerable, there is no need to reduce result quality by producing tentative tuples. In general, we see this approach only applicable to applications with very high data volume and very fast response times. In these applications short delays may be more important than result quality. This delay requirements have to be shorter than the time needed for recovery in an active standby approach which was already stated as negligible by the same work of the same group in [HBR+05]. Unfortunately, the work is not presenting an representative example application.

Recent work of the Aurora/Borealis group [HXCZ07] is moving towards the central ideas of this thesis which were fundamental part since the beginning of this work. The work in [HXCZ07] presents a fine-grained checkpointing approach that allows fragments of a continuous query graph to be recovered by different backup nodes. The work discovered that upstream backups do not efficiently support operators with large time windows because the complete time window has to be reprocessed in case of failures and therefore the group changed back to the passive standby approach with checkpointing. Moreover, the approach emphasizes to give a *no loss guarantee* and no longer focuses on degradation of query results.

Compared to the approach presented in our thesis, there are still major differences. Firstly, our approach is working at the most fine grained level of DSM operators and not on fragments of stream processes. We state that partitioning the stream process in appropriate fragments produces additional overhead which exceeding the gain of having less checkpointing overhead. Secondly, the work explicitly assume a cluster environment where server failures are rare events and therefore only single server failures are considered. As discussed in detail in Section 5.8.1, checkpointing for single failure scenarios does not need to consider the transfer state and therefore efficiency is easier to achieve. Finally, the work is focused on determining query fragments and efficient local scheduling of checkpoints which is not the primary focus of this thesis.

Research within TelegraphCQ in the subproject *Flux* [SHCF03] is also focusing on reliability aspects of DSM. Flux is an extension to Fjords in order to support distributed DSM processing in a shared-nothing network of nodes. In particular, Flux focuses on intra-operator parallelism also called *partitioned parallelism*. This means that multiple replicated instances of the same operator are executed in parallel at different nodes in the network. In order to share execution load, each of the operators is processing on a different time-window of the overall input streams at the same time. Finally, result streams of the different parallel operator instances are merged again and build the complete result. Obviously, this approach only works for DSM processing op-

erations that allow for partitioned execution over time. This implies that the current state of the operator is only depending on a limited time window of the input streams. After the end of each time window the processing starts from an empty state. This assumption is for example not valid for sliding window operations which never start over from an empty internal state again. Conceptually, Flux is implemented as an operator placed between a producer and consumer pair of DSM operators. Since each operator instance in the distributed DSM network considered in this work has multiple replicas running in parallel at different nodes, Flux is able to control which produced data stream elements of which producing instance are forwarded to which consuming instance. Moreover, Flux is able to perform load balancing not only within the available replicated operators but also the within the complete DSM network by migration of operator replicas with their states. This allows operator replicas to be moved from one node to another even during the processing within a time window. Further work [SHB04] on Flux is discussing how this mechanism can be extended for fault tolerance. With regard to the inherent constraint of having parallel running operator replicas, the failure handling follows an active standby approach. The proposed solution enables Flux to support load balancing and fault tolerance with minimal additional effort. Flux distinguishes between two reliability goals. Firstly, *loss-free* requires no data stream elements to be lost. Secondly *dup-free* requires even that no data stream elements are processed twice. The applied reliability algorithm is a combination between partitioned parallelism and active standby. The design goal of the approach is to achieve fast recovery without stalling DSM processing.

Compared to our approach, this active standby approach is not applicable to our intended application scenarios, where resources are limited and the overhead for reliability should be reduced. Moreover, the concept of partitioned parallelism is not applicable for general deterministic operators as supported by our DSM infrastructure. Finally, Flux is not considering multiple failure scenarios as considered within the work of this thesis.

In general, the presented related work in the DSM area differs in major points from the work presented in this thesis. Firstly, we have presented a consistent formal model of data stream management which is able to describe all forms of data streams, operators, stream processes within the data stream management system and their interactions on the outside world. Secondly, we have presented a formal approach to reliability of stream process execution at the level of operator instances. Building on this formal framework, we are able to define classes of reliability levels a data stream management system is able to provide. Contrarily to related work, we are not restricting operator behavior except for requiring determinism. Thirdly, we provide optimized

reliability algorithms based on the concept of operator migration. This is also contrarily to related work and allows to provide reliability at the lowest level of granulation within DSM. Reliability at operator level allows for maximum flexibility in failure handling. Moreover, the presented reliability algorithms have been proven formally based on the presented model and also empirical based on an evaluation within our real-world DSMS implementation.

8.3 Related Work in e-Health

In this section, we present application oriented research in e-Health with DSM demands. Research in this area is currently not using an integrated DSM infrastructure as proposed in this thesis although its application would be highly beneficial in order to provide reliability guarantees.

On the other hand, specific results presented in these projects are highly relevant as building blocks of a future integrated DSM infrastructure for healthcare (see Section 2.2). Moreover, the various projects underline the demand for general operator types as supported by the reliability model and algorithms presented in this thesis (see Chapter 5).

Furthermore, we describe commercial product solutions in the area of e-Health which would also benefit from the presented DSM infrastructure. The amount of already available commercial products in this area highly emphasizes the need for a reliable sensor data management infrastructure in the future.

8.3.1 Physiological Telemonitoring Projects

The *@HOME* Project at Fraunhofer [Sac02] allows for the monitoring of ECG, blood pressure, and oxygen saturation in conjunction with a programmable medication dispenser in order to improve the patient's compliance to medication. This project is focused on the use of wireless communication technologies and the applicability for telemonitoring applications.

The *Personal Health Monitoring System* (PHM) [Uni07a] is a project at University of Karlsruhe. Within this big project, various vital sensors and processing techniques for physiological data has been developed. Among them, a electrocardiographic telemetry system [WSN+00] has been developed which is remotely monitoring the electrical activity (ECG) of the heart 24 hours a day. In case of pathological heart beats, the ECG data is transmitted with an integrated GSM modem to the clinic's server. The patient also receives feedback from the caregiver (e.g., if an electrode has fallen off). This work focuses on ECG signal acquisition and processing. On-line feature ex-

traction and classification with respect to limitations of mobile devices (e.g., computation power and battery life) and also wireless communication technologies are discussed. In general research within the PHM project is focused on hardware, sensors, and signal processing issues in telemonitoring.

The *AMON* Project at the ETH Zurich [AWL+04] is focused on wearable devices for continuous medical monitoring of high-risk cardiac/respiratory patients. All sensors are integrated into a single wrist worn device, which also includes a computer for sensor data processing and a mobile phone unit for transmitting the results to a central server. The main issue in this work is the integration of all different units (signal processing, communication, user interface, and sensors) into a single wearable device. Other issues discussed within the project are sensor technologies, digital signal processing facing computation, power and memory constraints of embedded devices, and wireless communication technologies.

Cardenas et al. [CPC03] are working on management of streaming body sensor data for medical information systems. The described system uses the *Vivologic Life-Shirt* [Viv07], which is equipped with different body sensors and a database system for storing the sensor data, which is able to generate about one gigabyte of data per day. The project is focused on data management issues, like database schemes for storing streaming data, integration of stored streaming data with existing medical information, visualization and presentation of heterogeneous data.

Akogrimo [ako07] is an EU-funded project, which aims to integrate Grid technology and next generation mobile networks, and hence make Grid technology more appropriate to the mobile environment. The intended application scenarios of this project include smart-hospital, telemonitoring and emergency assistance. The proposed Grid infrastructure is able to deal with fluctuations in the environment like changing bandwidth, device capabilities, location. Furthermore the infrastructure can be immediately deployed in mobile environments.

Another EU-funded project in this area is MobiHealth [Mob03]. The project proposes a generic *body area network (BAN)* for wearable sensors. The focus of this project is on network technology, like providing plug and play sensor connectivity, security of sensor data, quality of service, and handover of roaming participants. Mobile devices like PDAs or smart-phones serve as mobile base units between the BAN and the outside world. The developed prototype application allows for monitoring, storage and wireless transmission (e.g., by using GPRS and UMTS technologies) of vital signs data coming from the patient BAN to the caregiver in a home telemonitoring setting. The successor project *Healthservice 24* [hea06] is focused on testing the feasibil-

ity of the prototype developed in MobiHealth in a real-life scenario and was finished successful.

The *Georgia Tech Wearable Motherboard* [GTW03] develops a wearable smart-shirt which incorporates sensors, like electrocardiogram (ECG) and temperature. The projects emphasize on development of an electronic garment woven in a normal shirt, which offers a data bus for different sensors that can also be included in the textile. The research focus of the project is on hardware issues of electronic garment and integrated sensors. The intended field of application is beside a battlefield scenario the telemonitoring of patients at home.

Assada et al. [ASR+03] are proposing a ring sensor for continuous monitoring of a patient's blood pressure in an unobtrusive way. This work is again focus on hardware issues of the ring sensors and necessary signal processing to derive correct measurements. The work also surveys the possible applications in the telemedical treatment of hypertension and heart failure patients at home.

MyHeart [myh06] is another research project funded by the EU. The focus of MyHeart is on preventing cardiovascular diseases by applying telemonitoring. The work in particular focuses on the telemonitoring of cardiovascular disease patients, where sensors integrated in clothing are used to monitor heart activity and physical activity. This project emphasizes on specialized sensors, e.g., innovative textile garment solutions, and device hardware, e.g., ultra-low power consuming electronics to be integrated in the garments, to allow for unobtrusive measurements. Moreover, telemonitoring specific issues and benefits for patients are evaluated.

Philips research in Aachen is also involved in the MyHeart projects, emphasize on research on smart textiles for telemonitoring. These textiles can be used for ECG measurement woven in underwear [MSS+04]. The *Healthcare Systems Architecture* (HSA) group of Philips Research Eindhoven [Phi07a] is doing research in the area of systems for personal health care, with a strong application focus. The research covers architecture and programming concepts [SWvS06] of applications in this area.

The EU-funded project *U-R-Safe* [urs05] builds up a telemonitoring environment for the elderly and chronical patients. The project develops a portable device which monitors continuously physiological signals (including heart activity, oxygen saturation, and fall detection) and is able to send an alarm to a medical center if abnormal values are detected. The technology issues addressed by this project cover sensor devices and wireless communication. Moreover application issues are investigated in this work.

Most investigated TM projects primarily focus on sensor or device hardware, communication technologies and signal processing [AWL+04, Sac02,

WSN+00]. We see a lack of infrastructure considerations in these projects as also stated by the *Wireless Wellness Monitor* [KPG03] project from the Technical Research Centre of Finland (VTT) [VTT07]. Therefore, we consider our proposed integrated DSM infrastructure as beneficial for the development of flexible, adaptable, and reliable TM applications in a distributed and heterogeneous network environment.

In the CodeBlue [Har07] project wireless vital sign sensors are designed to collect vital data and transmit them over a short-range wireless network to receiving devices. Further more a scalable software infrastructure for wireless medical devices is created. It provides routing, naming, discovery, and security for wireless medical sensors, and devices for monitoring and treating of patients. Thereby Code Blue can adapt to different network densities and different powerful wireless devices.

The *CustoMed* [JEZ+05] project is focused on wireless communication technologies, sensor devices and a medical vest. Intended application areas are the detection physical agitation of Alzheimer patients and monitoring forces and motion of the knees of post knee-surgery patients. Work within the projects explicitly emphasizes on of reliability and robustness [JDBS05] of the proposed monitoring hardware demanded by the applications.

Researchers at IBM Zurich [HNN04] have designed and built a platform, called *Personal Care Connect (PCC)*. The projects establish remote monitoring of patients by using a mobile personal hub interacting with patient sensors and building a gateway to the internet. The project is using existing sensors and devices from previous research at IBM, e.g., the Linux Watch [KIKT01] for the hub. The project is focusing on data-collection aspects of the system. Therefore interfaces to various Bluetooth-enabled medical devices have been developed. The researchers are working currently on an industry standard for medical device communication over Bluetooth.

Projects at the *Center for Pervasive Healthcare* [Per07] are focusing on aspects of infrastructure and application requirements in order to apply pervasive computing to healthcare. In particular, the *Java Context Awareness Framework (JCAF)* [J.E05] proposes an event-based infrastructure that offers the context-aware execution of services based on the idea of *activity based computing (ABC)*. The systems consists of a runtime middleware layer and an API to support the development of context-aware applications based on context information managed by the infrastructure.

Microsoft research is conducting the *HealthGear* project [OFM06]. HealthGear is developing a real-time wearable system for monitoring, visualizing and analyzing physiological signals. The application oriented research focuses on algorithms for automatic detection of sleep apnea by analyzing blood oxygen saturation and pulse while sleeping.

8.3.2 e-Inclusion and Ambient Assisted Living Projects

The *Nursebot* project [Uni07b] at University of Pittsburgh and the Carnegie Mellon University have devised fast algorithms for tracking variable numbers of people using laser range sensors mounted on mobile robots. The focus of the project are fast particle filters, which are statistical techniques well-suited for the localization and tracking of moving objects. The project has demonstrated to reliably estimate the right number of people and track their coordinates, even while the sensing robot moves about the environment.

The Speedy [DJRW03] fall detection is a very specific project focusing on the development of a wrist-worn device for elderly people to offer a wireless alarm system. The project emphasizes on the wrist-worn device and necessary signal processing to reliably detect fall events.

The Aware Home [ABE+02] is an interdisciplinary project at the Georgia Institute of Technology. The goal is to build a home that provides services to its residents that enhance their quality of life or help them to maintain independence as they age. The research is not only focused to fundamental technical issues but also social challenges imposed by applications in this field. A subpart of the project is focusing on measuring behavior of autistic children in order to support their education [HKT+04] by using a smart home infrastructure.

The *Medical Automation Research Center (MARC)* of University of Virginia has developed a smart in-home monitoring system [ADM+06]. The monitoring system has been implemented in simple low-cost sensor technology and it is possible to install the system in existing homes without much modifications. The system consists of motion detection sensors, controllable on-off switches, and fall detectors, to detect resident behavior and alarm conditions, e.g., fall. A pilot study conducted in 22 monitoring enabled-homes proofed user acceptance and improved quality of life.

Another smart home project at the *Center for Future Health* of University of Rochester is called *Smart Medical Home* [Uni07c]. The project establishes a laboratory for research consisting of a five-room house outfitted with infrared sensors, computers, bio-sensors, and video cameras for use by research teams to work with research subjects as they test their concepts and prototype products. Another project at the Center for Future Health is called *Milan* [HMCP04]. Milan proposes a middleware, designed to assist in the development of smart home applications.

Research at the Massachusetts Institute of Technology (MIT) is also dealing with smart home infrastructures for ambient assistive living. The MIT *PlaceLab* [MIT07] is a one-bedroom apartment with hundreds of sensing components are installed in nearly every part of the home. These sensors are be-

ing used to develop innovative user interface applications that help people to control easily their environment, save resources, remain mentally and physically active, and stay healthy. The sensors are also being used to monitor activity such that researchers can study how people react to the new infrastructure at home. As the Smart Medical Home the facility is managed as a multi-disciplinary shared scientific tool. Researchers from the MIT *Media Labs* [Int02] participate with PlaceLab and investigate within the *Changing Places* project how the home of the future can support and accommodate to offer a proactive environment for healthy living.

8.3.3 Wellness Monitoring Projects

Together with home automation systems, health monitoring devices form a smart environment, which monitors the lifestyle and generates feedback to improve the inhabitant's health. Representative for this field is the *Wireless Wellness Monitor* [KPG03], which is a prototype for supporting ubiquitous home applications. This project allows for automatic weight and activity monitoring, including circadian rhythm and fall detection. The focus is on easy and automatic measurements with a TV set and a PDA providing a clear user interface for intuitive feedback to keep up the user motivation.

The Intel Proactive Health [Int07] project is investigating how ubiquitous computing is able to support health and wellness of people in their daily life at home. The main emphasis is on development of proactive technologies that can anticipate the current needs of the person at home. At the project start, the research was focused on the telemonitoring scenario and ambient assisted living scenario, dealing with needs of patients suffering from cognitive decline, e.g., Alzheimer, cancer, or cardiovascular diseases. During the project, the research efforts have been expanded to focus on technologies to support also wellness applications, e.g., including physical fitness monitoring [CLB05]. They have designed and implemented a wearable multi-modal sensor device and developed classification techniques for the recognition of ten basic activities (like standing, sitting, or riding a bicycle) with high accuracy. Their multi-modal sensor board has seven sensors that measure the following quantities: audio, 3-axis acceleration, barometric pressure, temperature, humidity, compass heading and light level. The data can be transmitted wired or wireless to external computers (e.g., smartphone, PDA, or laptop).

HP Labs is conducting the *BioStream* project [BOHKVT06]. The researchers with signal-analysis and hardware architecture background are building an system that can collect patient physiological data (primarily electrocardiogram signals, blood pressure, temperature and weight) and analyze it for indications of problems or illness. The intended application scenario

is mainly in the wellness field, where individuals might constantly monitor their activity levels, making health and lifestyle choices accordingly – deciding, for instance, whether to end the day with a strenuous workout or just a brisk walk.

Research undertaken at the *Wearable Computing Lab* of ETH Zurich is focusing on embedded devices, sensors, and communication technology for wellness monitoring applications. For example, a dietary monitoring system [ASLT05] has been developed to detect whether a person is eating. Moreover also the type of food can be detected. The work was done as part of the previously mentioned MyHeart [myh06] project.

Closely related to the work at the Wearable Computing Lab is research done at the *Embedded Systems Lab* of the University of Passau where a toolbox [BKL06] for multi-modal context recognition application is developed. Contrarily to the more general emphasis on reliable data stream management in this thesis, this work is primarily focused on suitability for embedded devices and tailored to signal processing tasks applied to sensory data in order to derive the current context of the investigated object. Other work of this group is focusing on posture and gesture recognition and other motion analysis applications [BGL07]. Motion analysis is the basis for more complex applications in the area of wellness, ambient assisted living, and telemonitoring.

Research at the University of Alabama is developing a wireless personal monitor for detection of stress [JLR+03]. The wireless personal monitor is based on a *body area network* (BAN) in order to connect wearable sensors with a PDA. The project is focusing on the evaluation algorithms in order to detect a person's stress level and on wearable sensor hardware for measurement of vital signs like ECG, breathing rate, and brain activity (EEG). During the project the real-life applicability of system has been evaluated by military personnel performing the simulation of a helicopter emergency landing.

A research project at HP research labs [HL05] is working on a wearable wellness monitor which is able of continuously processing ECG and acceleration data. The system consists of off-shelf hardware for the ECG and acceleration sensors. A PDA device is the link between the sensors and the backend system. The system is able to extract activities, events and potentially important medical symptoms. The sensory data is transmitted wirelessly to the PDA via Bluetooth. The processed data is then forwarded to a back-end server for analysis using either a wireless internet connection, if available, or a cellular phone service. The project conducted experiments which are proofing the applicability of the system for activity monitoring, exercise monitoring and medical screening tests.

8.3.4 Commercial Products in e-Health and Wellness Monitoring

The Finnish healthcare technology company *IST Oy* has developed an automatic wrist-worn wireless monitoring device, called *Vivago WristCare* [IST07]. The device monitors the user's activity level and trigger alarms when necessary. The device communicates wirelessly with a base unit that analyzes the information and forwards the alarms through the telephone network. During the first four days of use, the unit adapts to the user's normal activity level by measuring acceleration, skin temperature and skin conductivity. If the device registers a significant change in the user's activity level, it automatically sends an alarm. The activity curve transmitted by the device can be used to support care activities with, for instance, the monitoring of sleep/wake rhythms. Researchers at VTT [KPS03] have evaluated the applicability of the Vivago system for the support of independent living of elderly in real-world settings.

Bodymedia [Bod07] is an US company that develops wearable body monitoring devices and services for collecting continuous physiological signs. Their most advanced wearable product is the *SenseWear armband*. A multi-sensor array integrated in the armband collects continuous data coming directly off the skin of the wearer's arm. The data can be transferred to a computer using either wireless communication or a standard USB cable. The armband is worn on the back of the upper right and measure body movement (with an 2-axis accelerometer), skin temperature, and skin conductivity. Out of this sensory data the system can calculate energy expenditure, durations of physical activity, number of steps, sleep/awake phases, and similar context information of the person. An additional event button allows users to timestamp and track specific incidents. The continuous data can be stored up to 7 days at a time within the device. Additional software is offered to analyze the gathered information. The intended field of application is in the area of wellness management, e.g., a user wants to pursue a daily workout plan in order to loose weight.

The Finish heart-monitor company *Polar Electro* and the sport company *Adidas* have formed a partnership to develop an integrated training monitoring system [Adi07]. They are integrating Polar monitoring equipment into Adidas clothing and footwear. Special fibres woven into Adidas sport shirts collaborate with Polar wireless heart-monitor device to eliminate the need for a separate chest strap in order to monitor heart rate. A special cavity in Adidas sport shoes allows to house a Polar stride sensor. Placing the sensor inside the shoe makes it easier to use, more comfortable, and more accurate

compared to top-of-shoe systems. Both sensors transmit their data wireless to a wrist-worn running computer for storage and analysis.

A similar product is offered by the computer company *Apple* in cooperation with the sport company *Nike*. Their *Nike + iPod Sport Kit* [App07] consists of a special Nike sport shoe with a cavity for a activity sensor. The activity sensor is wirelessly communicating with a *iPod nano*. For this reason the iPod nano has to be extended with a wireless receiver. An additional software running at the iPod is analyzing the sensory data and giving audiovisual feedback to the user about the training. Unfortunately, this system is not able to monitor heart activity.

The medical monitoring device company *Welch Allyn* is offering a wireless patient monitor, called *Micropaq* [Wel07]. The Micropaq is small wearable device that is continuously monitoring pulse, single or two lead electrocardiogram (ECG), and blood oxygen saturation. The Micropaq is able to connect wirelessly to a base station for further storage and analysis. The device has internal memory to store vital signals even when out of wireless connectivity. In addition, the device is able to show vital signal curves and data for analysis purposes at an integrated screen. The device is intended for ambulatory use and serious medical telemonitoring applications in hospital and out-of-hospital conditions.

The insulin pump manufacturer *Animas* is selling a completely noninvasive wrist-worn glucose monitor, the *GlucoWatch Biographer* [Ani07]. The device measures blood glucose level through the skin by using a replaceable pad that clips into the back of the GlucoWatch. The pad is glued to the skin to allow it to come in contact with a small electrical charge. This electrical charge brings glucose to the skin surface where an enzyme reaction, similar to that found in a standard glucose meter, generates electrons from which glucose levels can be closely estimated. The GlucoWatch measures glucose every 10 minutes (3 minutes of electrical stimulation, then 7 minutes of glucose measurement) and then these two measurements are averaged. The GlucoWatch is very helpful at showing patterns and trends in blood glucose level of diabetes patients, detecting after meal spiking, stopping nighttime low blood sugars, and sorting out the causes for morning highs. Problems with sweating, movement of patients, the need for an about 3h warm-up phase, and the fact that the hand of patient is not allowed to get in contact with water somewhat restrict this device from being a true continuous monitor.

The US pacemaker manufacturer *Medtronic* is developing telemedical products for both cardiac (pacemaker) patients and diabetes patients. The *MiniMed Paradigm* [Med07b] glucose monitoring is made up of two components, an invasive continuous glucose monitoring sensor, and an insulin pump. The systems transmits glucose readings every five minutes from the

glucose sensor to the insulin pump, which processes to 288 readings a day. The measurements are displayed on the insulin pump to allows patients to take immediate action in order to improve their glucose control. The continuous glucose sensor is a tiny electrode that is inserted under the skin, which can be applied by the patient with the help of a small insertion device. The electrode is typically replaced after three days of use. The insulin pump has a computer to manage the complex diabetes math for patients helping to avoid dangerous hypoglycemic episodes caused when too much insulin is delivered.

Another product is the *Medtronic CareLink Network* [Med07a] which is an Internet-based remote monitoring service for patients with Medtronic implanted cardiac devices and their clinics. The system consists of a base station which is connected to the caregiver and an antenna attached to it. The antenna is able to perform readings on the implant, which are forwarded to the caregiver in order to perform an online follow-up of the device. For this reason, the antenna has to be put on the chest over the implant. The implant is able to wirelessly connect to the antenna and deliver important information about itself and the patient's heart activity. In particular after an shock established by the implant and felt by the patient a follow-up check is necessary. Without the online follow-up the patient has to visit a hospital which imposes additional burden and time.

The US pacemaker manufacturer *Biotronik* [Bio07] is offering a similar telemonitoring solution for cardiac patients. In contrast to the Medtronic product, Biotronik is using a mobile phone to communicate with the cardiac implant. The implant sends periodically and in case of critical health conditions information via the mobile phone to the caregiver.

The CardioNet [Car07a] company provides an non-invasive, wearable, and wireless telemonitoring service for cardiac patients. A small sensors worn by the patient is continuously measuring the electrocardiogram (ECG). The sensor wirelessly connects to a PDA, which is continuously analyzing the heart activity. In case of abnormal health conditions the PDA is sending the ECG to the caregiver via the cellular network. An application of this system is to detect cardiac arrhythmias, which appear very infrequent and therefore are difficult to detect. Typically, the system is prescribed for a duration of up to 21 days. Medical research [RLS+07] has proven the improve in diagnosis of this system compared to traditional tools.

The *Georgia Tech Wearable Motherboard* [GTW03] is currently being manufactured for commercial use as *SmartShirt System* by *Sensatex* [Sen07]. The shirt is continuously monitoring the wearer's movement, heart rate, and respiration rate. A Bluetooth transmitter integrated in the shirt allows to wirelessly communicate the measurements to a base station. The shirt is very unobtrusive because the sensors are woven into the textile.

The company *VivoMetrics* has developed a sleeveless vest called *LifeShirt* [Viv07]. The shirt is made of stretch-material into which are sewn an array of physiologic sensors to monitor more than 30 vital signs, including electrocardiogram (ECG), blood pressure, respiration movement, body movement, and blood oxygen saturation. All sensors are by wire connected to an integrated data unit which is able to store the sensory data on a flash memory card. Roughly estimated the shirt is producing about one gigabyte of data per day. The analysis of the acquired data is done by a healthcare provider on regular checkups. Compared to the Sensatex product this shirt is more obtrusive.

The Dutch company TMSI has developed a mobile sensory data recorder, called Mobi [TMS07] system. A variety of physiological sensors, like e.g., ECG, respiration rate, accelerometers, can be attached by wire to the wearable device. The data recorder can record the different types of vital signs and is able to store the data on an internal flash memory. Additionally, the recorder is able to transmit the data to external devices (e.g. smartphone or PDA) using wireless communication.

The US company *Carematix* provides with their *Carematix Wellness System* [Car07b] a telemonitoring system for patients with chronic disease. The system monitors the vital parameters: blood pressure, pulse, weight, and blood glucose level. The monitoring devices are wirelessly connected to base station which forwards the sensory data to the caregiver. Unfortunately, the sensor devices are not wearable and monitoring is not continuous because measurements are performed by the patient at certain times. A very similar system from *CybernetMedical* called *MedStar* [Cyb07] offers telemonitoring of the same vital parameters for chronic diseases. The only difference to Carematix is that the sensors of this system are connected via cable links to the base station. Again the sensors are not wearable and measurements have to be initiated by the patient. Also the US technology company *Honeywell* [Hon07] offers a similar telemonitoring solution, with a wide variety of available sensor devices transmitting their measurements to a base station at the patients home which forwards the data to the healthcare provider. Again measurements are not continuous and the sensors are not wearable.

Also the Dutch technology company *Philips* offers a wide variety of telemonitoring devices [Phi07b] for treatment of chronic diseases. The offered wireless sensor devices are able to transmit measurements taken by the patient to a home base station which forwards the sensory data to the healthcare provider. Again, we see the issue that the offered sensors are not wearable and the measurements have to be initiated by the patient.

A very interesting and flexible product in the are of pervasive computing with potential applications in wellness management is the *AySystem* [Sie07]

from the German technology company *Siemens*. The AySystem is a mobile, wearable device of about the size of a common mobile phone. The device is GSM/GPRS enabled and includes an integrated acceleration sensor, a temperature sensor, a sound sensor, as well as a loudspeaker. The device includes a web-server that enables remote users to query the state of the device over the Internet. Moreover, the device is able to send text messages if certain programmable thresholds of sensory values are exceeded. In addition, notifications to the local user are given. Finally, the device allows to initiate a two-way voice communication. Moreover, a snap-on system offers the possibility to attach additional sensor hardware like a GPS receiver.

9

Conclusion & Outlook

In this thesis, we have addressed the reliable data stream management in a distributed environment including also mobile devices. We summarize our contributions and outline topics for future work.

9.1 Contribution

In this dissertation, we have presented a novel information management infrastructure for reliable data stream and process management. A vision for the future is that such an infrastructure is able to cope with workflow and DSM processing demands of modern distributed applications mainly coming from the pervasive and ubiquitous computing area. An example application is telemonitoring in healthcare. As described in the thesis, telemonitoring applications greatly benefit from a reliable DSM and process management infrastructure supporting also mobile devices. In addition, the thesis gives an overview of healthcare application research and issues targeted by application research. Investigations of various healthcare projects in these fields have shown a lack of infrastructure supporting the generation of telemonitoring applications.

Moreover, we have emphasized in this thesis on achieving a very high degree of reliability for these applications although they are running in a volatile distributed setting where failures are very likely to happen. Nevertheless, the infrastructure is able to use available resources efficiently and compensating failures seamlessly and transparently for the application. In order to have a formal foundation for our work, we have introduced a model for DSM where data stream operators are distributed across a loosely coupled network of hosts. Based on the DSM model, we have identified failures that

are likely to occur in a DSMS during runtime. Furthermore, as one major contribution of the thesis, we have formally defined reliability levels of DSM, based on input/output behavior of the DSMS which is considered as black box seen from the outside world. The reliability levels, in turn, allow for a precise characterization of the consistency a DSMS is able to provide at runtime. The three dimensions along which the reliability levels are defined address i.) limited-loss and lossless DSM, ii.) limited-delay and delay-free DSM, and iii.) intra-stream and inter-stream order preservation.

Based on this formalism, we have developed a set of suitable and efficient reliability algorithms for DSM to allow for efficient operator migration in case of failures or overload situations in Peer-to-Peer fashion. In this thesis, we present and evaluate *Efficient Coordinated Operator Checkpointing* (ECOC). ECOC provides consistent operator checkpoints with low overhead. In order to allow the migration of stateful data stream operators in case of failures, a recent checkpoint of the state is needed. We have formally proven that ECOC meets the lossless reliability level. In particular, the emphasis was to reduce network overhead which is highly relevant for mobile environments. ECOC allows to reduce the drawbacks of passive standby approach presented in [HBR+05], i.e., high runtime and recovery overhead. Further optimizations of ECOC support real world stream processing scenarios having complex distributed operator graphs including joins, splits and even cycles.

As opposed to to most research in the field of DSM [HBR+05, BBMS05, BBC+04, SHB04, C+03], the presented reliability strategies for DSM in this thesis are performing at the *data stream operator* level in a distributed environment. Reliability at the operator level and not at the level of a whole stream processing node allows for a fine grained load redistribution in case of failures or overload situations by operator migration.

The ideas presented in this thesis are implemented in a DSM enabled successor of the existing process management infrastructure OSIRIS [SWSS04, SST+05, SWSS03], which is called *OSIRIS-SE* (Stream Enabled) [BS07, BSS06, BSS05]. Since the original OSIRIS has been implemented in proprietary C++ limited to Windows platforms, we have implemented OSIRIS-SE in Java in order to allow for platform independence, i.e., to support mobile and embedded device platforms that offer an appropriate Java virtual machine. Moreover, we have integrated the support for *reliable execution* of continuously running DSM processes or *stream processes* in the Java-based OSIRIS-SE infrastructure. Essentially, stream processes have to be continuously fed with incoming sensor data. An important requirement is also to provide an easy-to-use graphical interface that can be used by non-programmers (e.g., physicians or care personnel) to design new or to revise existing patient- and/or disease-specific stream processes. For this reason, the OSIRIS process

design tool *O'Grape* [WSN+03] has also been extended to support the design of stream processes.

The experimental part of the thesis is emphasizing on two aspects. One aspect is to provide a demonstrator implementing a simplified real world telemonitoring scenario by incorporation of a set of real world sensors and mobile devices. This demonstrator has been shown on various scientific events [BS07, SB06] in order to give a practical impression on the requirements and tasks imposed by a complex DSM application.

The more important second part of the evaluations is to analyze performance and overhead of the proposed reliability techniques. Since the majority of DSM applications are inherently distributed and executed in a heterogenous environment, we have performed the evaluations on both server hardware and mobile devices. For these evaluations, we still use real sensory data for DSM processing. Moreover, the evaluations cover both the performance during the normal (failure-free) runtime of the system and the performance during the phase when recovering from failures for stream processes of various different topology. For the failure analysis, we have even analyzed a multi-failure scenario.

Finally, the thorough evaluations presented in this thesis, have shown a significant reduction of the overhead for checkpointing by still keeping the desired lossless reliability level. In particular, our coordinated ECOC approach and its optimizations are performing significantly better than the standard passive standby approach with respect to network overhead. Network overhead is the major drawback of the uncoordinated passive standby approach. In addition, we have shown that our approach does not cause higher CPU or memory overhead both during normal runtime and recovery/catchup phase. In general, the evaluations have shown that our proposed DSM reliability algorithms are efficient and affordable with reasonable overhead even in mobile environments. Major limitations are not given due to the applied reliability algorithms on mobile devices. Still limitations on mobile devices are given by inefficient Java implementations, inefficient wireless network drivers, and operating system issues regarding multitasking applications (e.g., starvation of tasks).

9.2 Outlook to Future Research

One interesting problem of future research is the combination of alternative stream processing branches with transparent failure handling. Alternative stream processing branches allow to bypass an failed operator instance which has no more available provider in the network. For example, Fred's base sta-

tion at home is going down which was hosting a complex heart analysis operator. Unfortunately, the only available node is Fred's PDA which is not able to host this complex operator instance. Nevertheless, Fred's PDA is able to perform a less sophisticated analysis of the heart activity by using an operator instance of a different type. Fred's caregiver has modeled within the stream process that this operator type can be used as alternative processing branch. A naive approach, can be to stop the current stream process and establish a new stream process by using the alternative branch. The drawback of this approach is that all state information aggregated before the failure is lost even though stored in a checkpoint backup. As topic of further research, the infrastructure should be able to perform operator migration between different operator types and allow an operator specific *checkpoint conversion service* to convert the checkpoint according to the new operator type. In this case, the new operator instance is not starting DSM processing from scratch. Instead, processing results aggregated in the operator state of the failed instance can be reused to avoid loss of relevant information.

Another research topic is trigger operator migration also for load balancing to optimize the load distribution within the DSM infrastructure. This mechanism can actively relieve highly loaded nodes before load gets too high and leads to a complete congestion. In this case, long delays will already trigger operator migration in our current DSM infrastructure as failure handling. But the drawback of this is that the congestion has already happened and needs to be worked off with additional delays for a longer period of time. If we can prevent overload and resulting congestion by operator migration for load balancing, the overall environment is better utilized and causes less delays. Of course, an important aspect in this approach is to find a reasonable tradeoff between performance gain due to operator redistribution and the resulting overhead and delays caused by the operator migration itself. Applying heuristics about migration costs, operator execution costs, and data stream rates can be a first attempt.

In general, when applying a more sophisticated load management and having more flexibility with alternative stream processing branches, we extend the self-healing and self-adaption capabilities of the system. In this scenario, also dynamic deployment of operators is an interesting extension. Dynamic deployment will allow to install an operator type dynamically on nodes with free resources.

When analyzing the mobile environment, we still see that the JVM imposes a significant overhead compared to native code implementation. An interesting research task will be to implement a lightweight native version of the DSM infrastructure OSIRIS-SE in order to evaluate the actual overhead of Java. Unfortunately, leaving the Java world has major drawbacks

opposed to the Java "compile once, run everywhere" philosophy. For example, the need to manage various native versions which need to be compatible among each other when working together in a heterogenous environment. Also for dynamic deployment various operator type implementations need to be managed because of incompatible runtime environments.

Bibliography

[A+03] K. Aberer et al. *Advanced Peer-to-Peer Networking: The P-Grid System and its Applications*. *PIK Journal - Praxis der Informationsverarbeitung und Kommunikation, Special Issue on P2P Systems*, 2003.

[AAB99] J. Allaire, P. Allaire, and M. Baloh. *Swallow frequency device: A method of swallow detection.* In *Proc. of the RESNA '99 Annual Conference*, pages 8 – 10, Long Beach, CA, USA, June 1999.

[AAB+05] D. J. Abadi, Y. Ahmad, M. Balazinska, U. Cetintemel, M. Cherniack, J.-H. Hwang, W. Lindner, A. Maskey, A. Rasin, E. Ryvkina, N. Tatbul, Y. Xing, and S. B. Zdonik. *The Design of the Borealis Stream Processing Engine*. In *Proc. of CIDR Conference*, pages 277–289, January 2005.

[ABE+02] G. A. Abowd, I. Bobick, E. Essa, R. Mynatt, and W. Rogers. *The Aware Home: Developing Technologies for Successful Aging*. In *Proc. of AAAI Workshop and Automation as a Care Giver*, Alberta, Canada, 2002.

[ABKM01] D. G. Andersen, H. Balakrishnan, F. Kaashoek, and R. Morris. *Resilient Overlay Networks*. In *Proc of Symposium on Operating System Principles (SOSP)*, pages 131–145, Banff, Canada, October 2001.

[ABW03] A. Arasu, S. Babu, and J. Widom. *CQL: A Language for Continuous Queries over Streams and Relations.* In *Database Programming Languages (DBPL) Workshop*, pages 1–19, Potsdam, Germany, September 2003.

[Adi07] Adidas, Polar. *The AdiStar Fusion Training System*. `http://www.adidas-polar.com`, 2007.

[ADM+06] M. Alwan, S. Dalal, D. Mack, S. Kell, B. Turner, J. Leachtenauer, and R. Felder. *Impact of monitoring technology in assisted living: outcome pilot*. *IEEE Transactions on Information Technology in Biomedicine*, 10(1):192 – 198, 2006.

[AH00] R. Avnur and J. M. Hellerstein. *Eddies: Continuously Adaptive Query Processing*. In *Proc. of the SIGMOD Conference, May 16-18, 2000, Dallas, Texas, USA*, pages 261–272, May 2000.

[ako07] *Access to Knowledge through the Grid in a Mobile World (Akogrimo)*. http://www.mobilegrids.org, 2007.

[Ame07] American Heart Association. *Heart Disease and Stroke Statistics 2007 Update*. *American Heart Association*, 2007.

[Ana03] *Analog Devices iMEMS Accelerometers*. www.analog.com/technology/mems/accelerometers, 2003.

[Ani07] Animas Technologies. *The Glucowatch G2 Biographer*. http://www.glucowatch.com, 2007.

[App07] Apple Inc. *Nike and iPod Sports Pack*. http://www.apple.com/ipod/nike/gear.html, 2007.

[ASLT05] O. Amft, M. Stäger, P. Lukowicz, and G. Tröster. *Analysis of Chewing Sounds for Dietary Monitoring*. In *Proc. of International Conference on Ubiquitous Computing*, pages 56–72, September 2005.

[ASR+03] H. H. Asada, P. Shaltis, A. Reisner, S. Rhee, and R. C. Hutchinson. *Mobile Monitoring with Wearable Photoplethysmographic Biosensors*. *IEEE EMB Magazine*, 22(3):28–40, 2003.

[AWL+04] U. Anliker, J. A. Ward, P. Lukowicz, G. Tröster, F. Dolveck, M. Baer, F. Keita, E. B. Schenker, F. Catarsi, L. Coluccini, A. Belardinelli, D. Shklarski, M. Alon, E. Hirt, R. Schmid, and M. Vuskovic. *AMON: A Wearable Multiparameter Medical Monitoring and Alert System*. *IEEE Transactions on Information Technology in Biomedicine*, 8(4):415–427, 2004.

[Axi] *Apache Web Services Project - Axis*. http://ws.apache.org/axis.

[B+01] R. Braumandl et al. *ObjectGlobe: Ubiquitous query processing on the Internet*. *VLDB Journal*, 10(1):48–71, 2001.

[Bar81] J. Bartlett. *A NonStop Kernel*. In *Proc. of ACM Symposium on Operating Systems Principles*, pages 22–29, Asilomar, CA, USA, 1981.

[BBC+04] H. Balakrishnan, M. Balazinska, D. Carney, U. Cetintemel, M. Cherniack, C. Convey, E. F. Galvez, J. Salz, M. Stonebraker, N. Tatbul, R. Tibbetts, and S. B. Zdonik. *Retrospective on Aurora*. *VLDB Journal*, 13(4):370–383, 2004.

[BBD+02] B. Babcock, S. Babu, M. Datar, R. Motwani, and J. Widom. *Models and Issues in Data Stream Systems*. In *Proc. of PODS Conf.*, pages 1–16, Madison, WI, USA, 2002.

[BBGA03] D. Benatar, M. Bondmass, J. Ghitelman, and B. Avitall. *Outcomes of Chronic Heart Failure*. *Arch. Intern Med. 2003*, 163:347–352, 2003.

[BBMS05] M. Balazinska, H. Balakrishnan, S. Madden, and M. Stonebraker. *Fault-Tolerance in the Borealis Distributed Stream Processing System*. In *Proc. of ACM SIGMOD Conf.*, pages 13–24, Baltimore, MD, USA, June 2005.

[BGH86] J. Bartlett, J. Gray, and B. Horst. *Fault Tolerance in Tandem Computer Systems*. Technical Report TR 86.2, Tandem, 1986.

[BGL07] M. Barry, A. Grünerbl, and P. Lukowicz. *Wearable joint-angle measurement with modulated magnetic field from l/c oscillators*. In *Proc. of International Workshop on Wearable and Implantable Body Sensor Networks (BSN)*, March 2007.

[Bio07] Biotronik. *The Biotronik Home Monitoring Service*. `http://www.biotronik-healthservices.com`, 2007.

[BKL06] D. Bannach, K. Kunze, and P. Lukowicz. *Distributed modular toolbox for multi-modal context recognition*. In *Proc. of International Conference on Architecture of Computing Systems (ARCS)*, pages 99–113, March 2006.

[BMM+04] S. Babu, R. Motwani, K. Munagala, I. Nishizawa, and J. Widom. *Adaptive ordering of pipelined stream filters*. In *Proc of SIGMOD Conference*, pages 407–418, June 2004.

[Bod07] BodyMedia. *BodyMediaő Body Monitoring System*. `http://www.bodymedia.com/products/bodymedia.jsp`, 2007.

[BOHKVT06] A. Bar-Or, J. Healey, L. Kontothanassis, and J. Van Thong. *BioStream: A System Architecture for Real-Time Processing of Physiological Signals*. In *Proc. of the IEEE Engineering in Medicine and Biology Society Conference (EMBS)*, pages 3101–3104, 2006.

[BQF99] R. Baldoni, F. Quaglia, and P. Fornara. *An Index-Based Checkpointing Algorithm for Autonomous Distributed Systems*. *IEEE Trans. Parallel Distrib. Syst.*, 10(2):181–192, 1999.

[BS07] G. Brettlecker and H. Schuldt. *The OSIRIS-SE (stream-enabled) infrastructure for reliable data stream management on mobile devices*. In *Proc. of SIGMOD Conf.*, pages 1097–1099, June 2007.

[BSS04] G. Brettlecker, H. Schuldt, and R. Schatz. *Hyperdatabases for Peer–to–Peer Data Stream Processing*. In *Proc. of ICWS Conf.*, pages 358–366, San Diego, CA, USA, 2004.

[BSS05] G. Brettlecker, H. Schuldt, and H.-J. Schek. *Towards Reliable Data Stream Processing with OSIRIS-SE*. In *Proc. of BTW Conf.*, pages 405–414, Karlsruhe, Germany, March 2005.

[BSS06] G. Brettlecker, H. Schuldt, and H.-J. Schek. *Efficient and Coordinated Checkpointing for Reliable Distributed Data Stream Management*. In *Proc. of ADBIS Conf.*, pages 296–312, October 2006.

[BW01] S. Babu and J. Widom. *Continuous Queries over Data Streams*. *SIGMOD Record*, 30(3):109–120, 2001.

[C+02] D. Carney et al. *Monitoring Streams - A New Class of Data Management Applications*. In *Proc. of VLDB Conf.*, pages 215–226, Hong Kong, China, 2002.

[C+03] S. Chandrasekaran et al. *TelegraphCQ: Continuous Dataflow Processing for an Uncertain World*. In *Proc. of CIDR Conf.*, Asilomar, USA, 2003.

[Car07a] CardioNet. *The CardioNet System*. `http://www.cardionet.com`, 2007.

[Car07b] Carematix. *The Carematix Wellness System*. `http://www.carematix.com`, 2007.

[CBB+03] M. Cherniack, H. Balakrishnan, M. Balazinska, D. Carney, U. Cetintemel, Y. Xing, and S. Zdonik. *Scalable Distributed Stream Processing*. In *Proc. of CIDR Conference*, Asilomar, CA, USA, 2003.

[CDTW00] J. Chen, D. DeWitt, F. Tian, and Y. Wang. *NiagaraCQ: A Scalable Continuous Query System for Internet Databases*. In *Proc. of SIGMOD Conf.*, pages 379–390, Dallas, TX, USA, 2000.

[CFP+06] K. Chintalapudi, T. Fu, J. Paek, N. Kothari, S. Rangwala, J. Caffrey, R. Govindan, E. Johnson, and S. Masri. *Monitoring Civil Structures with a Wireless Sensor Network*. *Internet Computing*, 10(2):26 – 34, 2006.

[CHK+06] M. Cammert, C. Heinz, J. Kramer, T. Riemenschneider, M. Schwarzkopf, B. Seeger, and A. Zeiss. *Stream Processing in Production-to-Business Software*. In *Proc. of the 22nd International Conference on Data Engineering (ICDE'06)*, page 168, 2006.

[CHKS03] M. Cammert, C. Heinz, J. Krämer, and B. Seeger. *Datenströme im Kontext des Verkehrsmanagements*. In *Mobilität und Informationssysteme*, Zurich, Switzerland, October 2003.

[CKP+05] R. Cheng, B. Kao, S. Prabhakar, A. Kwan, and Y. Tu. *Adaptive stream filters for entity-based queries with non-value tolerance*. In *Proc of VLDB Conference*, pages 37–48, September 2005.

[CL85] K. M. Chandy and L. Lamport. *Distributed snapshots: determining global states of distributed systems*. *ACM Trans. Comput. Syst.*, 3(1):63–75, 1985.

[CLB05] T. Choudhury, J. Lester, and G. Borriello. *Assessing Wellness by Sensing Everyday Activities and Interactions*. In *Proc. of HCI Challenges in Health Assessment*, Portland, OR, USA, April 2005.

[CLD08] K. Casey, A. Lim, and G. Dozier. *A Sensor Network Architecture for Tsunami Detection and Response*. *International Journal of Distributed Sensor Networks*, 4(1):28 – 43, 2008.

[Cod70] E. F. Codd. *A Relational Model of Data for Large Shared Data Banks*. *Communications of the ACM*, 13(6):377–387, 1970.

[CPC03] A. Cardenas, R. Pon, and R. Cameron. *Management of Streaming Body Sensor Data for Medical Information Systems*. In *Int. Conf. on Mathematics and Engineering Techniques in Medicine and Biological Sciences*, pages 186–191, Las Vegas, NV, USA, June 2003.

[Cyb07] Cybernet. *The Cybernet MedStar System*. `http://www.cybernetmedical.com`, 2007.

[Der] *The Apache Derby Project*. http://db.apache.org/derby/.

[Dey01] A. K. Dey. *Understanding and Using Context*. *Personal Ubiquitous Computing*, 5(1):4–7, 2001.

[DG92] D. DeWitt and J. Gray. *Parallel Database Systems: The Future of High Performance Database Systems. Commun. ACM*, 35(6):85–98, 1992.

[DJRW03] T. Degen, H. Jaeckel, M. Rufer, and S. Wyss. *SPEEDY: A Fall Detector in a Wrist Watch*. In *Proc. of International Symposium on Wearable Computers (ISWC)*, page 184, Los Alamitos, CA, USA, 2003.

[DK76] F. DeRemer and H. H. Kron. *Programming-in-the-Large Versus Programming-in-the-Small. IEEE Transactions on Software Engineering*, 2(2):80–86, 1976.

[DMM+02] V. Dialani, S. Miles, L. Moreau, D. D. Roure, and M. Luck. *Transparent Fault Tolerance for Web Services Based Architectures*. In *Proc. of the 8th International Euro-Par Conference on Parallel Processing*, pages 889–898, August 2002.

[DPSB01] K. Dansky, L. Palmer, D. Shea, and K. Bowles. *Cost Analysis of Telehomecare*. *Telemedicine Journal and e-Health*, 7(3), 2001.

[DTAK07] M. Duller, R. Tamosevicius, G. Alonso, and D. Kossmann. *XTream: personal data streams*. In *Proc. of SIGMOD Conference*, pages 1088–1090, June 2007.

[EAWJ02] E. Elnozahy, L. Alvisi, Y. Wang, and D. Johnson. *A survey of rollback-recovery protocols in message-passing systems*. *ACM Comput. Surv.*, 34(3):375–408, 2002.

[Eur05] Eurostat. *Population Projections 2004-2050*. `http://epp.eurostat.ec.europa.eu/pls/portal/docs/PAGE/PGP_PRD_CAT_PREREL/PGE_CAT_PREREL_YEAR_2005/PGE_CAT_PREREL_YEAR_2005_MONTH_04/3-08042005-EN-AP.PDF`, 2005.

[Eur06] European Commission. *EU IST Strategic Objectives*. `http://cordis.europa.eu/ist/activities/activities.htm`, 2006.

[FJL+97] S. Floyd, V. Jacobson, C.-G. Liu, S. McCanne, and L. Zhang. *A reliable multicast framework for light-weight sessions and application level framing*. *IEEE/ACM Transactions on Networks*, 5(6):784–803, 1997.

[FK05] P. M. Fischer and D. Kossmann. *Batched Processing for Information Filters*. In *Proc. of ICDE Conference*, pages 902–913, April 2005.

[GHM02] C. Giraldo, S. Helal, and W. Mann. *mPCA Ű A Mobile Patient Care-Giving Assistant for Alzheimer Patients*. In *Int. Workshop on Ubiquitous Computing for Cognitive Aids*, Göteborg, Sweden, October 2002.

[GL03] B. Gedik and L. Liu. *PeerCQ:A Decentralized and Self-Configuring Peer-to-Peer Information Monitoring System*. In *Proc. of Distributed Computing Systems Conf.*, pages 490–499, Providence, RI, USA, 2003.

[GMK88] H. Garcia-Molina and B. Kogan. *Achieving High Availability in Distributed Databases*. *IEEE Trans. Softw. Eng.*, 14(7):886–896, 1988.

[GMP90] H. Garcia-Molina and C. Polyzios. *Two epoch algorithms for disaster recovery*. In *Proc. of VLDB*, pages 222–230, 1990.

[GO03] L. Golab and M. T. Özsu. *Issues in Data Stream Management*. *ACM SIGMOD Record*, 32(2), 2003.

[Gra78] J. Gray. *Notes on Data Base Operating Systems*. In *Operating Systems, An Advanced Course*, pages 393–481. Springer-Verlag, 1978.

[GTW03] *GTWM – The Georgia Tech Wearable Motherboard: The Intelligent Garment for the 21st Century*. `http://www.smartshirt.gatech.edu`, 2003.

[HA99] C. Hagen and G. Alonso. *Highly Available Process Support Systems: Implementing Backup Mechanisms*. In *Proc. of the 18$^{\text{th}}$ Symposium on Reliable Distributed Systems (SRDS'99)*, pages 112–121, Lausanne, Switzerland, Oct 1999.

[HAHK02] R. Haux, E. Ammenwerth, W. Herzog, and P. Knaup. *Health care in the information society. A prognosis for the year 2013. Int. Journal of Medical Informatics*, 66:3–21, 2002.

[Har07] Harvard University, Cambridge, MA. *CodeBlue: Wireless Sensor Networks for Medical Care*. `http://www.eecs.harvard.edu/~{}mdw/proj/codeblue`, 2007.

[HBR+05] J. Hwang, M. Balazinska, A. Rasin, U. Cetintemel, M. Stonebraker, and S. Zdonik. *High Availability Algorithms for Distributed Stream Processing*. In *Proc. of ICDE Conference*, pages 779 – 790, April 2005.

[hea06] *The Healthservice 24 Project*. `http://www.healthservice24.com`, 2006.

[HJR+98] S. Harsdorf, M. Janssen, R. Reuter, B. Wachowicz, and R. Willkomm. *Lidar as part of an ROV-based sensor network for the detection of chemical pollutants on the seafloor*. In *Proc. of OCEANS Conference*, pages 1250–1253, Nice, France, 1998.

[HK03] K. Hildrum and J. Kubiatowicz. *Asymptotically Efficient Approaches to Fault-Tolerance in Peer-to-Peer Networks.* In *Proc. of International Symposium on Distributed Computing*, pages 321–336, 2003.

[HKT+04] G. R. Hayes, J. A. Kientz, K. N. Truong, D. R. White, G. D. Abowd, and T. Pering. *Designing Capture Applications to Support the Education of Children with Autism.* In *Proc. of International Conference on Ubiquitous Computing (Ubicomp)*, pages 161–178, September 2004.

[HL05] J. Healey and B. Logan. *Wearable wellness monitoring using ECG and accelerometer data*. In *Proc. of. IEEE Symposium on Wearable Computers (ISWC)*, pages 220–221, Osaka, Japan, October 2005.

[HMCP04] W. Heinzelman, A. Murphy, H. Carvalho, and M. Perillo. *Middleware to Support Sensor Network Applications*. *IEEE Network Magazine*, 18(1):5–14, 2004.

[HNN04] D. Husemann, C. Narayanaswa, and M. Nidd. *Personal Mobile Hub*. In *Proc. of International Symposium on Wearable Computers (ISWC)*, pages 85–91, Washington, DC, USA, 2004.

[Hon07] Honeywell. *Honeywell HomMed*. `http://www.hommed.com`, 2007.

[HSW+00a] J. Hill, R. Szewczyk, A. Woo, S. Hollar, D. Culler, and K. Pister. *System Architecture Directions for Networked Sensors*. *SIGPLAN Notices*, 35(11):93–104, 2000.

[HSW+00b] J. Hill, R. Szewczyk, A. Woo, S. Hollar, D. Culler, and K. Pister. *System architecture directions for networked sensors*. *ACM SIGPLAN Notices*, 35(11):93–104, 2000.

[HWMB04] M. A. Humphrey, G. S. Wasson, M. M. Morgan, and N. Beekwilder. *An Early Evaluation of WSRF and WS-Notification via WSRF.NET*. In *Proc. of International Workshop on Grid Computing (GRID)*, pages 172–181, 2004.

[HWS04] R. H., R. Winter, and J. Schiller. *A partition detection system for mobile ad-hoc networks*. In *Proc. of Conference on Sensor and Ad Hoc Communications and Networks*, pages 489–497, October 2004.

[HXCZ07] J.-H. Hwang, Y. Xing, U. Cetintemel, and S. Zdonik. *A Cooperative, Self-Configuring High-Availability Solution for Stream Processing*. In *Proc. of ICDE Conference*, pages 176–185, April 2007.

[IBM04] IBM. *IBM Websphere MQ Workflow Concepts and Architecture Version 3.5*, 2004.

[Int02] S. Intille. *Designing a Home of the Future*. *IEEE Pervasive Computing*, 1(2):76–82, 2002.

[Int07] Intel Research. *Proactive Health*. `http://www.intel.com/research/prohealth`, 2007.

[IST07] IST Oy. *VIVAGO Wrist Care*. `http://www.istsec.fi/vivago.php?lang=eng`, 2007.

[J+03] A. Jones et al. *Final Report of the CRA Conference on Grand Research Challenges in Information Systems*. Technical report, Computing Research Association, 2003. `http://www.cra.org/reports/gc.systems.pdf`.

[J2S] *Java Platform, Standard Edition 6*. http://java.sun.com/javase/.

[JDBS05] R. Jafari, F. Dabiri, P. Brisk, and M. Sarrafzadeh. *Adaptive and fault tolerant medical vest for life-critical medical monitoring*. In *In Proc. of Symposium on Applied Computing (SAC)*, pages 272–279, 2005.

[J.E05] B. J.E. *The Java Context Awareness Framework (JCAF) - A Service Infrastructure and Programming Framework for Context Aware Applications*. In *Proc. of International Conference on Pervasive Computing (PERVASIVE)*, pages 98–115, Munich, Germany, 2005.

[Jet] *Jetty 100% Java HTTP and J2EE Servlet Container*. http://jetty.mortbay.org.

[JEZ+05] R. Jafari, A. Encarnacao, A. Zahoory, F. Dabiri, H. Noshadi, and M. Sarrafzadeh. *Wireless sensor networks for health monitoring*. In *Proc. of International Conference on Mobile and Ubiquitous Systems (MobiQuitous)*, pages 479–481, San Diego, CA, USA, 2005.

[JLR+03] E. Jovanov, A. O. Lords, D. Raskovic, P. G. Cox, R. Adhami, and F. Andrasik. *Stress Monitoring Using a Distributed Wireless Intelligent Sensor System*. *IEEE EMB Magazine*, 22(3):66–73, 2003.

[JXT] *The Project JXTA*. https://jxta.dev.java.net.

[KAGM96] M. Kamath, G. Alonso, R. Günthör, and C. Mohan. *Providing High Availability in Very Large Worklflow Management Systems*. In *Proc. of EDBT*, pages 427–442, Avignon, France, 1996.

[KIKT01] N. Kamijoh, T. Inoue, K. Kishimoto, and K. Tamagawa. *Linux Watch: Hardware Platform for Wearable Computing Research*. In *Proc. of Pacific Rim Conference on Multimedia (PCM)*, pages 1–7, 2001.

[KIW05] Z. Kalbarczyk, R. K. Iyer, and L. Wang. *Application Fault Tolerance with Armor Middleware*. *IEEE Internet Computing*, 9(2):28–37, 2005.

[KPG03] I. Korhonen, J. Pärkkä, and M. v. Gils. *Health Monitoring in the Home of the Future*. *IEEE EMB Magazine*, 22(3):66–73, 2003.

[KPS03] I. Korhonen, P. Paavilainen, and A. Särelä. *Application of ubiquitous computing technologies for support of independent living of the elderly in real life settings*. In *Proc. of the International Workshop on Ubiquitous Computing for Pervasive Healthcare Applications*, Seattle, WA, USA, October 2003.

[KS04] J. Krämer and B. Seeger. *PIPES: A Public Infrastructure for Processing and Exploring streams*. In *Proc. of ACM SIGMOD Conf.*, pages 925–926, 2004.

[KSS04] J. Kleinberg, M. Sandler, and A. Slivkins. *Network failure detection and graph connectivity*. In *Proc. of Symposium On Discrete Algorithms (SODA)*, pages 76–85, 2004.

[KWA+03] R. Kumar, M. Wolenetz, B. Agarwalla, J. Shin, P. Hutto, A. Paul, and U. Ramachandran. *DFuse: A Framework for Distributed Data Fusion*. In *Proc. of SensSys Conf.*, pages 114–125, Los Angeles, CA, USA, 2003.

[KWSB02] U. Kunzmann, G. v. Wagner, J. Schöchlin, and A. Bolz. *Parameter Extraction of the ECG-Signal in Real-Time*. *Biomed. Technik 47*, 2:875–878, 2002.

[KYC+06] J. Krämer, Y. Yang, M. Cammert, B. Seeger, and D. Papadias. *Dynamic Plan Migration for Snapshot-Equivalent Continuous Queries in Data Stream Systems.* In *Proc of EDBT Workshops*, pages 497–516, March 2006.

[Ley05] F. Leymann. *The (Service) Bus: Services Penetrate Everyday Life.* In *ICSOC*, pages 12–20, Amsterdam, Netherlands, Dec 2005.

[LL73] C. L. Liu and J. W. Layland. *Scheduling Algorithms for Multiprogramming in a Hard-Real-Time Environment*. *Journal of the ACM*, 20(1):46–61, 1973.

[LL04] J.-W. Liu and J.-J. Le. *WSRF-Based Computing Framework of Distributed Data Stream Queries.* In *Proc. of International Conference on Grid and Cooperative Computing (GCC)*, pages 951–954, 2004.

[LP96] J. Lin and S. Paul. *RMTP: a reliable multicast transport protocol*. In *Proc. of the IEEE INFOCOM Conference*, pages 1414–1424, March 1996.

[LPT99] L. Liu, C. Pu, and W. Tang. *Continual Queries for Internet Scale Event-Driven Information Delivery*. *IEEE Transactions on Knowledge and Data Engineering*, 11(4):610–628, 1999.

[LTBL97] M. Litzkow, T. Tannenbaum, J. Basney, and M. Livny. *Checkpoint and migration of unix processes in the condor distributed*

processing system. Technical Report TR 1346, University of Wisconsin-Madison, 1997.

[LZJ+05] B. Liu, Y. Zhu, M. Jbantova, B. Momberger, and E. A. Rundensteiner. *A Dynamically Adaptive Distributed System for Processing Complex Continuous Queries.* In *Proc of VLDB Conference*, pages 1338–1341, September 2005.

[Mat01] F. Mattern. *The Vision and Technical Foundations of Ubiquitous Computing*. *Upgrade*, 2(5):2–6, October 2001.

[Med07a] Medtronic. *Medtronic CareLink Network for Pacemakers*. `http://www.medtronic.com/physician/carelink/pacemakers.html`, 2007.

[Med07b] Medtronic. *The MiniMed Paradigm REAL-Time Insulin Pump and Continuous Glucose Monitoring System*. `http://www.minimed.com/products/insulinpumps`, 2007.

[MF02] S. Madden and M. J. Franklin. *Fjording the Stream: An Architecture for Queries Over Streaming Sensor Data*. In *Proc. of International Conference on Data Engineering (ICDE)*, pages 555–566, February 2002.

[MFHH02] S. R. Madden, M. J. Franklin, J. M. Hellerstein, and W. Hong. *Tag: A Tiny Aggregation Service for Ad-hoc Sensor Networks*. In *OSDI 2002*, 2002.

[Mica] *Microsoft Windows Server System - Microsoft SQL Server*. http://www.microsoft.com/sql/.

[Micb] *Microsoft Visual Studio*. http://msdn.microsoft.com/vstudio/.

[MIT07] MIT Department of Architecture. *The PlaceLab*. `http://architecture.mit.edu/house_n/placelab.html`, 2007.

[Mob03] *The MobiHealth Project*. `http://www.mobihealth.org`, 2003.

[Moy98] J. Moy. *RFC-2328: Open Shortest Path First(OSPF)*. http://rfc.sunsite.dk/rfc/rfc2328.html, 1998.

[MRD+07] R. Mueller, J. S. Rellermeyer, M. Duller, G. Alonso, and D. Kossmann. *A dynamic and flexible sensor network platform*. In

SIGMOD '07: Proceedings of the 2007 ACM SIGMOD international conference on Management of data, pages 1085–1087, June 2007.

[MSS+04] J. Muhlsteff, O. Such, R. Schmidt, M. Perkuhn, H. Reiter, J. Lauter, J. Thijs, G. Musch, and M. Harris. *Wearable approach for continuous ECG - and activity patient-monitoring*. In *Proc. of International Conference on Engineering in Medicine and Biology Society (IEMBS)*, pages 2184–2187, September 2004.

[MST04] M. Mlivoncic, C. Schuler, and C. Türker. *Hyperdatabase Infrastructure for Management and Search of Multimedia Collections.* In *Proc. of DELOS Workshop: Digital Library Architectures*, pages 25–36, S. Margherita di Pula, Cagliari, Italy, 2004.

[myh06] *The MyHeart Project*. `http://www.hitech-projects.com/euprojects/myheart`, 2006.

[MyS] *MySQL AB - MySQL*. http://www.mysql.com.

[NAS00] NASA's Jet Propulsion Laboratory California. *Wearable Sensor Patches for Physiological Monitoring*. `http://www.nasatech.com/Briefs/Feb00/NPO20651.html`, 2000.

[NGYS00] B. Natarajan, A. Gokhale, S. Yajnik, and D. C. Schmidt. *DOORS: Towards High-Performance Fault Tolerant CORBA*. In *International Symposium on Distributed Objects and Applications*, pages 39–40, September 2000.

[NHEB01] G. A. Nichols, T. A. Hillier, J. R. Erbey, and J. B. Brown. *Congestive heart failure in type 2 diabetes: prevalence, incidence, and risk factors*. *Diabetes Care*, 24(9):1614–1619, 9 2001.

[NW03] M. Naor and U. Wieder. *A Simple Fault Tolerant Distributed Hash Table.* In *Proc. of IPTPS International Workshop on Peer-to-Peer Systems*, pages 88–97, 2003.

[OFM06] N. Oliver and F. Flores-Mangas. *HealthGear: A Real-time Wearable System for Monitoring and Analyzing Physiological Signals*. In *Proc. of International Workshop on Wearable and Implantable Body Sensor Networks (BSN)*, pages 61–64, Boston, MA, USA, 2006.

[Per07] *Centre for Pervasive Healthcare, University of Aarhus, Denmark*. http://www.pervasivehealthcare.dk, 2007.

[Phi07a] Philips. *Philips Healthcare Systems Architecture*. http://www.extra.research.philips.com/swa/cluster_hcs.html, 2007.

[Phi07b] Philips. *Philips Telemedicine Solutions*. http://www.medical.philips.com/in/products/telemonitoring, 2007.

[Phy] *PhysioNet the research resource for complex physiologic signals*. http://www.physionet.org/.

[Pos81] J. Postel. *RFC-793: Transmission Control Protocol*. http://rfc.sunsite.dk/rfc/rfc793.html, 1981.

[RBS+02] M. Rogers, D. Buchan, D. Small, C. Steward, and B. Krenzer. *Telemedicine improves diagnosis of essential hypertension compared with usual care*. *Journal of Telemed. and Telecare*, 8:344–349, 2002.

[Rek95] Y. Rekhter. *RFC-1771: Border Gateway Protocol 4 (BGP-4)*. http://rfc.sunsite.dk/rfc/rfc1771.html, 1995.

[RG07] T. M. R. G. Ramakrishna Gummadi, Nupur Kothari. *Declarative Failure Recovery for Sensor Networks*. In *Proc. of the Conference on Aspect-Oriented Software Development (AOSD)*, Vancouver, Canada, March 2007.

[RKH03] P. Ryan, R. Kobb, and P. Hilsen. *Making the Right Connection: Matching Patients to Technology*. *Telemedicine Journal and e-Health*, 9(1), 2003.

[RLS+07] S. A. Rothman, J. C. Laughlin, J. Seltzer, J. S. Walia, R. I. Baman, S. Y. Siouffi, R. M. Sangrigoli, and P. R. Kowey. *The Diagnosis of Cardiac Arrhythmias: A Prospective Multi-Center Randomized Study Comparing Mobile Cardiac Outpatient Telemetry versus Standard Loop Event Monitoring*. *Journal of Cardiovascular Electrophysiology*, 18(3):1–7, January 2007.

[RSB+01] M. Rogers, D. Small, D. Buchan, C. Butch, C. Stewart, B. Krenzer, and H. Husovsky. *Home Monitoring Service Improves Mean Arterial Pressure in Patients with Essential Hypertension*. *Annals of Internal Medicine*, 134(11):1024–1032, 2001.

[RW97] D. S. Rosenblum and A. L. Wolf. *A Design Framework for Internet-Scale Event Observation and Notification*. In *Proc. of European Software Engineering Conference (ESEC)*, pages 344–360, 1997.

[SABS02] H. Schuldt, G. Alonso, C. Beeri, and H.-J. Schek. *Atomicity and Isolation for Transactional Processes*. *ACM Transactions on Database Systems*, 27(1):63–116, 2002.

[Sac02] I. Sachpazidis. *@HOME: A modular telemedicine system*. In *Workshop on Mobile Computing in Medicine*, Heidelberg, Germany, 2002.

[SB06] H.-J. Schek and G. Brettlecker. *Research Issues in Pervasive Information Management for Health Monitoring and e-Inclusion*. In *Proc. of ECEH Conf.*, pages 11–12, October 2006.

[SBG+00] H.-J. Schek, K. Böhm, T. Grabs, U. Röhm, H. Schuldt, and R. Weber. *Hyperdatabases*. In *Proc. of WISE Conf.*, pages 28–40, Hong Kong, China, 2000.

[Sen07] Sensatex. *The SmartShirt System*. `http://www.sensatex.com`, 2007.

[SFG+02] J. Saia, A. Fiat, S. Gribble, A. Karlin, and S. Saroiu. *Dynamically Fault-Tolerant Content Addressable Networks*. In *Proc. of IPTPS International Workshop on Peer-to-Peer Systems*, pages 270–279, 2002.

[SH98] M. Sullivan and A. Heybey. *Tribeca: A System for Managing Large Databases of Network Traffic*. In *Proc. of USENIX Conf.*, New Orleans, LA, USA, 1998.

[SH03] F. Stann and J. Heidemann. *RMST: Reliable Data Transport in Sensor Networks*. In *Proc. of the Workshop on Sensor Net Protocols and Applications*, pages 102–112, Anchorage, Alaska, USA, May 2003.

[SHB04] M. Shah, J. Hellerstein, and E. Brewer. *High Available, Fault-Tolerant, Parallel Dataflows*. In *Proc. of ACM SIGMOD Conf.*, pages 827–838, 2004.

[SHCF03] M. Shah, J. Hellerstein, S. Chandrasekaran, and M. Franklin. *Flux: An Adaptive Partitioning Operator for Continuous Query Systems*. In *Proc. of ICDE Conf.*, Bangalore, India, 2003.

[Sie07] Siemens AG. *Siemens AySystem*. `http://pia.khe.siemens.com/index.aspx?nr=14834`, 2007.

[SKK04] B. Stegmaier, R. Kuntschke, and A. Kemper. *StreamGlobe: adaptive query processing and optimization in streaming P2P environments*. In *Proc. of the International Workshop on Data Management for Sensor Networks*, pages 88–97, 2004.

[SL07] D. Savio and T. Ludwig. *Smart Carpet: A Footstep Tracking Interface*. In *Proc. of Conference on Advanced Information Networking and Applications Workshops*, pages 754–760, Washington, DC, USA, 2007.

[SLR96] P. Seshadri, M. Livny, and R. Ramakrishnan. *The Design and Implementation of a Sequence Database System*. In *Proc. of VLDB Conf.*, pages 99–110, 1996.

[SMLN+03] I. Stoica, R. Morris, D. Liben-Nowell, D. R. Karger, M. F. Kaashoek, F. Dabek, and H. Balakrishnan. *Chord: a scalable peer-to-peer lookup protocol for internet applications. IEEE/ACM Transactions on Networking*, 11(1):17–32, 2003.

[Sop05] Sophie Petersen and Viv Peto and Mike Rayner and Jose Leal and Ramon Luengo-Fernandez and Alastair Gray. *The European Cardiovascular Disease Statistics*. *European Heart Network*, 2005.

[Sri06] N. Sridhar. *Decentralized Local Failure Detection in Dynamic Distributed Systems*. In *Proc. of Symposium on Reliable Distributed Systems (SRDS)*, pages 143–154, 2006.

[SSP+01] L. S., W. S., J. P., M. K., and L. E. *Compliance and effectiveness of 1 year's home telemonitoring. The report of a pilot study of patients with chronic heart failure*. *European Journal of Heart Failure*, 3(6):723–730, 2001.

[SSSW02] H.-J. Schek, H. Schuldt, C. Schuler, and R. Weber. *Infrastructure for Information Spaces*. In *Proc. of ADBIS Conf.*, pages 22–36, Bratislava, Slovakia, 2002.

[SST+05] C. Schuler, H. Schuldt, C. Türker, R. Weber, and H.-J. Schek. *Peer-to-Peer Execution of (Transactional) Processes*. *International Journal of Cooperative Information Systems (IJCIS)*, 14(4):377–405, 2005.

[SSW02a] H.-J. Schek, H. Schuldt, and R. Weber. *Hyperdatabases – Infrastructure for the Information Space*. In *Proc. of VDB Conf.*, pages 1–15, Brisbane, Australia, 2002.

[SSW+02b] S. Shea, J. Starren, R. Weinstock, P. Knudson, J. Teresi, D. Holmes, W. Palmas, L. Field, R. Goland, C. Tuck, G. Hripcsak, L. Capps, and D. Liss. *Columbia University's Informatics for Diabetes Education and Telemedicine (IDEATel) Project: Rationale and Design*. *JAMIA*, 9(1):49–62, 2002.

[SWSS03] C. Schuler, R. Weber, H. Schuldt, and H.-J. Schek. *Peer–to–Peer Process Execution with OSIRIS*. In *Proc. of ICSOC Conf.*, pages 483–498, Trento, Italy, 2003.

[SWSS04] C. Schuler, R. Weber, H. Schuldt, and H.-J. Schek. *Scalable Peer–to–Peer Process Management – The OSIRIS Approach*. In *Proc. of ICWS Conf.*, pages 26–34, San Diego, CA, USA, 2004.

[SWvS06] W. Stut, F. Wartena, and M. van Steen. *A Distributed Shared Data Space for Personal Health Systems*. In *Proc. of International Congress of the European Federation for Medical Informatics (MIE)*, pages 57–64, August 2006.

[The03] The Globus Alliance. *The Globus Toolkit Version 3*. `http://www-unix.globus.org/toolkit/`, 2003.

[TMS07] TMS International. *The Mobi system*. `http://www.tmsi.com`, 2007.

[TS01] A. S. Tanenbaum and M. Steen. *Distributed Systems: Principles and Paradigms*. Prentice Hall PTR, 2001.

[Uni07a] University of Karlsruhe. *The Personal Health Monitoring System*. `http://www.itiv.uni-karlsruhe.de/opencms/opencms/en/research/projects/mst_phm/`, 2007.

[Uni07b] University of Pittsburg and Carnegie Mellon University. *The Nursebot Project*. `http://www.cs.cmu.edu/~nursebot`, 2007.

[Uni07c] University of Rochester. *SMART MEDICAL HOME RESEARCH LABORATORY*. `http://www.futurehealth.rochester.edu/smart_home`, 2007.

[urs05] *Universal Remote Signal Acquisition For hEalth (U-R-SAFE)*. http://ursafe.tesa.prd.fr, 2005.

[Viv07] VivoMetrics. *The LifeShirt System*. http://www.vivometrics.com, 2007.

[VTT07] *Technical Research Centre of Finland, Wellness and Healthcare*. http://www.vtt.fi, 2007.

[WALJ+06] G. Werner-Allen, K. Lorincz, J. Johnson, J. Lees, and M. Welsh. *Fidelity and yield in a volcano monitoring sensor network*. In *Proc. of the symposium on operating systems design and implementation*, pages 381–396, 2006.

[WC79] E. W.A.H. and Z. C. *A single scan algorithm for QRS-detection and feature extraction*. *Journal of Computers in Cardiology*, 6:37–42, 1979.

[WCK02] C.-Y. Wan, A. T. Campbell, and L. Krishnamurthy. *PSFQ: a reliable transport protocol for wireless sensor networks*. In *Proc. of the workshop on Wireless Sensor Networks and Applications (WSNA)*, pages 1–11, September 2002.

[WCP+99] S. Warren, R. L. Craft, R. C. Parks, L. K. Gallagher, R. J. Garcia, and D. R. Funkhouser. *A Proposed Information Architecture for Telehealth System Interoperability*. In *Toward an Electronic Patient Record Š99*, Orange County Convention Center, Orlando, FL, 1999.

[Wel07] WelchAllyn. *The Micropaq Wireless Patient Monitor*. http://www.monitoring.welchallyn.com/products/wireless/micropaq.asp, 2007.

[WeM] *WebSphere Everyplace Micro Environment*. http://www-306.ibm.com/software/wireless/weme/.

[WLLP01] B. Warneke, M. Last, B. Liebowitz, and K. Pister. *Smart Dust: Communicating with a Cubic-Millimeter Computer*. *IEEE Computer Magazine*, 34(1):44–51, 2001.

[WSN+00] G. v. Wagner, S. Schubert, L. Ngambia, C. Morgenstern, and A. Bolz. *Concept for an Event-Triggered Electrocardiographic Telemetry-System using GSM for supervision of cardiac patients*. In *Conf. Rec. of the 2000 AFCEA/IEEE EUROCOMM*, Munich, Germany, 2000.

[WSN+03] R. Weber, C. Schuler, P. Neukomm, H. Schuldt, and H.-J. Schek. *Web Service Composition with OGrape and OSIRIS*. In *Proc. of VLDB Conf.*, Berlin, Germany, 2003.

[YDHH06] Y. Yang, D. Divan, R. Harley, and T. Habetler. *Power line sensornet - a new concept for power grid monitoring*. *IEEE Power Engineering Society General Meeting*, pages 8 pp.–, 2006.

[YG03] Y. Yao and J. Gehrke. *Query Processing for Sensor Networks*. In *Proc. of CIDR Conf.*, Asilomar, CA, USA, 2003.

[YSM07] L. Yamamoto, D. Schreckling, and T. Meyer. *Self-Replicating and Self-Modifying Programs in Fraglets*. In *Proc of Conference on Bio-Inspired Models of Network, Information, and Computing Systems (BIONETICS)*, Budapest, Hungary, December 2007.

[YTP05] Y. X. Y. Tu, M. Hefeeda and S. Prabhakar. *Control-based Quality Adaptation in Data Stream Management Systems*. In *Proc of DEXA Conference*, pages 746–755, August 2005.

[ZKJ02] B. Y. Zhao, J. Kubiatowicz, and A. D. Joseph. *Tapestry: a fault-tolerant wide-area application infrastructure. Computer Communication Review*, 32(1):81, 2002.

[ZRH04] Y. Zhu, E. A. Rundensteiner, and G. T. Heineman. *Dynamic Plan Migration for Continuous Queries Over Data Streams*. In *Proc. of SIGMOD Conference*, pages 431–442, Paris, France, June 2004.

Curriculum Vitae

Gert Brettlecker

November 19, 1974	Born in Innsbruck, Austria Son of Elsa and Werner Brettlecker
1981–1985	Primary school, Innsbruck
1985–1989	Secondary school, Gymnasium, Innsbruck
1989–1994	Höhere Technische Bundeslehranstalt, Innsbruck
1994	Matura
1999–2000	Military service, Austrian army
1994–2002	Study of Electrical Engineering, Vienna University of Technology
2002	Diplom-Ingenieur (equals MSc) in Electrical Engineering, Vienna University of Technology
2002–2006	Research assistant in the group of Prof. H.-J. Schek, Institute for Information Systems, UMIT Hall in Tyrol.
2006–2008	Research assistant in the group of Prof. Heiko Schuldt, Database and Information Systems, University of Basel.