3

Research Opportunities in Business Process Technology

Improvements in business process technology are clearly demanded by the strategic need of today's enterprises to become more flexible in the sense of reactiveness to the more and more rapidly changing business environments.

IT systems in an enterprise are seldom designed from scratch, they evolve along new demands over the years, so that system landscapes [156] emerge. So, in practice, the issue of making enterprise IT more flexible is about fostering the flow of information by enterprise application integration efforts – protection of investment is the rationale for this pragmatic approach. In research, we are free to look at the problem with a fresh look – actually from scratch. Now is the time to systematically analyze the needs, driving forces and benefits of business process technology by looking onto the plethora of concrete business process management products and their features from a conceptual viewpoint. In the terminology of current software engineering technology and software processes it is the task to define a component-based platform for business processes that unifies modeling, construction, operations and maintenance of business process software. Such a goal is not only a mental exercise for researchers, also industry is foreseeing such new integrative products that go beyond current business process management suites and the term business process platform has been coined for them.

With respect to business process technology we have identified two potential fields of research, i.e., executable specification and components. We delve into these topics in Sects. 3.2 resp. 3.3. They are the major topics of this book. The topics address the improvement of business process technology independent from specific business functionality. It is also interesting to invest research into the dimension of specific business functionalities – again from an enterprise application integration viewpoint. In particular, we currently still see a gap between IT support for administration and production processes in manufacturing enterprises. It should not be forgotten that excellence in production is a foundation of today's businesses [155].

Enterprises react to technological trends to stay competitive. In the past, more and more concrete applications and systems were introduced to address

D. Draheim, *Business Process Technology: A Unified View on Business Processes, Workflows and Enterprise Applications*, DOI 10.1007/978-3-642-01588-5_3,
© Springer-Verlag Berlin Heidelberg 2010

more and more problems, starting from basic numerical control systems to today's manufacturing execution systems [237, 202], from basic accounting systems to today's enterprise resource planning systems, from basic reporting capabilities to today's analytical processing systems, from basic electronic data interchange to today's logistics management systems. The potential of automation is still huge in modern enterprises. In general, a focus on the mere administration side of businesses is too narrow. There are current initiatives like MESA and ISA-95 that address the integration of business processes and production processes. We delve into this topic in Sect. 3.5. Having the production process in mind can prevent us from making flaws in the design of future business process platforms from the outset.

3.1 Business Process Platforms

For us, the term business process platform does not stand for some kind of improved integration of standard components into business process management suites. For us, the term business process platform has two aspects, i.e., executable specification and component-based architecture. The point is that the creation of a high-level specification mechanism for business processes comes first, i.e., we believe that the existence of such a notion of executable specifications is a precondition for a working component architecture and not vice versa. This viewpoint differs crucially from how commercial product vendors approach the problem of creating a next generation business process platform. The approach we see at the vendor's side follows a certain tradition of enterprise application integration that has ruled the design of business process technology in the past, i.e., the understanding of enterprise application integration as a step-wise improvement of the information flows in an enterprise system landscape with the objective to touch existing systems as little as possible or even not to touch the systems at all instead of radically refactoring, reengineering or even reconstructing the systems. Therefore the promises of component architecture for business process management suites usually do not go beyond the simplification and better support for hooking realizing software parts together, i.e., the provision of a plug-in architecture. However, the languages for the construction of software parts are not the focus of improvement – they are considered as given.

In research, on the contrary, we have the chance to approach things more fundamentally. We can consider languages and mechanisms for the construction of software parts and those for gluing software parts together as a whole. Furthermore, we can view systems from outside, i.e., the viewpoint of system specification from an end-user's viewpoint – we are not restricted to the viewpoint of the given virtual machine defined by a concrete programming language. This means that there is a potential to design a domain-specific language for the specification and construction of business process software – and if the job is done right, specification and construction are actually the same, which is expressed by the notion of executable specification. We believe that only an appropriate notion of compositionality of such an executable specification mechanism yields a truly powerful component architecture for business process software.

By the way, it is strange that we have already seen fourth generation (4GL) languages like RPG (Report Generator) for the midrange computer AS/400 that have been designed for the implementation of business logic and the typical form-based dialogues of enterprise applications but today's advertised new business process management suites all rely on third generation (3GL) languages for the implementation of the software components they hook together. And actually we see migration projects in practice from systems that are implemented in such domain-specific technologies to platforms that are based on a current object-oriented programming language. Superficially, such migration projects are sometimes motivated by the desire to migrate to object-oriented technology. However, the argument only works if you take for granted the superiority of object-oriented technologies over older ones. However, the maturity of domain-specific 4GL languages like the aforementioned RPG should not be underestimated. There are usually more concrete reasons for such migration projects. One reason could be simply the better availability of programmers for a newer non-proprietary programming language. Another could be the insight that the higher costs of a current system in terms of total costs of ownership actually do not pay off, because the non-functional requirements on the system are actually not so high that they justify the costs. Another reason could be the following from many classical legacy problem scenarios: the functionality of the system has to be made available in the setting of a newer technology – most probably web technology – and it has been estimated that re-implementing the system is the cheaper or at least less risky alternative to wrapping and embedding the system.

We said that the analysis of the driving forces of business process management on the problem side and the features of current business process technology on the solution side is necessary preparatory work for the design of next generation business process platforms. We believe that a scientific analysis must not be misled by the promises of any software engineering metaphor. In particular, a mere programming language level discussion can easily miss the point here. A system should be considered as an entirety of software, middleware and hardware – the issue in question is how good these components are orchestrated and in how far their design is streamlined by overall objectives and design rationales. We believe that such considerations are important to reach technology independency eventually. For example, RPG shows its value as part of the holistically designed AS/400 system [325, 93] – today known as i-series, 'System i5' or 'System i' – with its co-designed and co-constructed operating system, database management system and virtual machine system, i.e., OS/400 – today known as i5/OS – DB2 and TIMI (Technology Independent Machine Interface) respectively. The reason for the system's robustness is in this case that the hardware and software components are designed for each other following crosscutting design principles.

3.2 Executable Specification of Business Processes

We have seen steady efforts to make business process specifications executable, both in academia [134] and industry [247, 272]. There are two non-mutual and converging communities that foster this trend, i.e., the business process modeling community, e.g., [265], and the workflow management community [164]. Business processes are an issue in enterprises, e.g., [151, 150], even without executable semantics of processes.

Workflow control has its origins in concrete technologies for computer-supported collaborative work (CSCW) based on document processing like Palo Alto's OfficeTalk [192] in the 1970s or Polymer [232] in the 1980s, on the one hand, and in more general rapid development frameworks based on a worklist paradigm like FlowMark [218], on the other hand. A lot of today's commercial business process management suites [247] actually started as workflow management products.

3.2.1 Means of Business Process Automation

In principle, the target of executable business process specification can be approached top-down, by hooking business process modeling tools with executable systems, or bottom-up by enriching workflow engines. However, the gap remains; there is no canonical mapping between the components that are under the control of workflow technology and the entities addressed by business process modeling. The view of business process modeling is rather a global one, i.e., the net of business activities and exchanged information entities. The view of workflow control, on the other hand, is a local one, looking at the human computer interaction and having a concrete worklist paradigm at hand for processing workflows. We believe that the gap between business process modeling and workflow control should be systematically investigated. As a quick gain, it is possible to exploit the results of such investigations as best practices in practical business process projects. In the long run the results can help in the unification of both levels and the design of an advanced business process management suite.

A step in bridging the gap between business process modeling and business process management can be done by an investigation of advanced role-model concepts from a workflow patterns perspective. There has been a rigorous discussion of workflow patterns in the workflow community [1] that helped in the investigation and analytical comparison of existing workflow technology. This workflow pattern discussion has already been broadened [308] by the consideration of workflow resources, i.e., different users. User and user role models are at the heart of the workflow paradigm. Considering users and roles can bring a human-computer interaction viewpoint to the discussion of workflow patterns refining the otherwise global, i.e., observational viewpoint of an overall action flow. The findings of such human-computer interaction focused investigations can be exploited in the definition of an executable specification language for business processes. For example, the definition of the single user session of a submit/response-style system as typed, bipartite state machine can serve as a basis [89]. Here, the human-computer interaction is form-oriented – it consists of an ongoing interchange of report presentations and form submissions. In this setting it is possible to understand the notion of worklist as an interaction pattern in single user session scenarios and to proceed by generalizing the defined semantic apparatus to a form-oriented workflow definition language.

We believe that a future business process platform should allow for the executable specification of workflows and dialogues. In such a platform there will no longer be any artificial distinction between the workflow states and the states of the dialogues that bridge the workflow states. This means, system dialogues and workflows are unified [109]. An immediate major benefit of this platform is that important BPM techniques like business process monitoring and business process simulation are no longer artificially restricted to some coarse-grained workflow states, they become pervasive. Furthermore, the business logic is partitioned naturally into services of appropriate granularity this way. The decision as to which parts of the supported business process is subject to workflow technology and which parts make up the dialogues is orthogonal to the specification of the business process, i.e., a posteriori. The definition can be changed allowing for a yet unseen degree of flexibility in business process specification.

3.2.2 Inter-Organizational Business Process Automation

It is a further challenge to integrate business process platforms with approaches for inter-organizational supply chain management and extended supply chain management [348, 347]. This challenge has a technical and a conceptual, i.e., business relevant, aspect. The technical challenge is about distributed deployment. If the component architecture of a business process platform is done properly support for distributed deployment can be added easily to the platform. As we will argue in the course of the book we consider a component architecture as appropriate for a business process platforms if it allows for the unrestricted decomposition of software at the outermost level of process specifications.

Support for distributed deployment is good also for intra-organizational purposes; anyhow, with the correct exploitation of virtualization technology [65, 137] there is the chance that the differences between software architecture [132] and deployment architecture vanish – in particular, in these days of emerging virtualization technologies for commodity servers like Xen [78] or VmWare [331]. But as we have mentioned, there is also a business related challenge of inter-organizational distribution of business processes and this is the challenge of negotiating responsibilities. It will be interesting to see which kind of information technology can actually support and add value to this issue.

3.2.3 Executable Specification Communities

The synonyms for executable specification range from old ones like automatic programming [279] to today's model-driven architecture [248, 40, 90, 91].

> "*In short, automatic programming always has been an euphemism for programming with a higher-level language than was then available to the programmer. Research in automatic programming is simply research in the implementation of higher-level programming languages.*" [279]

Executable specification is about gaining a new level of abstraction in the description of systems that have an operational semantics. Such endeavors are typically domain-specific, i.e., phenomena in the program design that occur often are identified and become new constructs of a new virtual machine. Therefore it is fair to say that domain-specific languages [349, 79] and even generative programming [67, 98, 99, 100, 86] are also in the realm of executable specification. Actually, it is a common misconception about model-driven architecture that this approach gains a higher-level of abstraction for general purpose program system construction. On the contrary, the research community in model-driven architecture is very well aware of the fact that the real work to be done is in defining domain-specific modeling languages that then can be exploited further to generate systems. The model-driven architecture approach is rather about setting the stage for the systematic definition of modeling languages and a kind of standardization of tool support for these definition efforts, i.e., we think it can be understood somehow as a disciplined approach to meta case tools [159, 160, 115, 246, 212, 197].

3.3 Component-Based Development

The notion of software component has been discussed as early as the NATO software engineering conference [256, 51]. Components are about code composition. But people associate more than composition mechanisms with the concept of components. Actually, there are lots of abstraction and composition mechanisms available in programming languages – routines, procedures, modules [277], objects. However, the discussion of components goes beyond the design of composition mechanisms, it also goes beyond the discussion on how to decompose systems [278] for maximal robustness or reuse. However, different communities put different emphasis in their discussion on component technology, so the concept comes with different flavors. There is a sub industry aspect, an infrastructure aspect, and a large system construction aspect.

As we will see in due course these aspects are not mutually exclusive. We need to discuss these three aspects in Sects. 3.3.1 to 3.3.3 in order to gain a better understanding of current and future trends in component-orientation for business process management suites.

The notion of business process platform as currently used and foreseen by industry has component-orientation as a crucial asset. Here, the sub industry of components is dominating, expressing the vision that next generation business process management suites are prepared for gluing together ready-made business logic components. The development of an appropriate component-model for business processes is driven this way. Our approach to component-orientation is more fundamental. Our concept of component is really just composition and composability. The usefulness of composability is beyond doubt. What we are seeking is a notion of composition which makes that the composition of arbitrary business process specification immediately yields a valid new business process specification. It is our conviction that just "yet another plug-in component architecture" that targets easier deployment of business process implementations will not bring the promised new quality of business process technology. Our targeted notion of component-orientation for business processes is indivisibly connected to the design of a next-level specification language for business processes – we consider an appropriate component architecture of business process platforms rather as a by-product of robust design efforts of a high-level specification language and not as an independent asset.

3.3.1 Sub Industry Aspect of Component Technology

One important aspect of component technology is that they are about establishing software sub industries. This is probably the earliest usage of the term component [241]. This means the term component is used for the division of programming efforts at the level of software houses. In [241] input-output conversion, two and three dimensional geometry, text processing, and storage management are given as examples for possible components supplied by specialized software houses to other software houses. With respect to this sub industry aspect frameworks and application programming interfaces (APIs) clearly are components.

Still, the sub industry aspect is often considered as the defining aspect of component technology. However, the perception of the topic has changed. In motivations of component technology research ordinary, i.e., existing application programming interfaces are usually not mentioned but rather domain-specific business logic components. Here the specialization is along industrial sectors or concrete businesses.

3.3.2 Infrastructure Aspect of Component Technology

In practice, concrete component technologies are about adding technical value to a specific technological domain by creating an infrastructure for it – we therefore use the term infrastructure aspect. These technological domains crosscut industrial sectors. Examples of technological domains are the field of visual programming, the field of distributed object computing and the wide field of enterprise computing. Component technologies address one such domain with a combination of foundational software services and tools. One very ubiquitous view of component technology is to see it as an extension of object-oriented programming technology [335].

For the sake of completeness we list the usual examples. SUN's JavaBeans are a component technology for visual programming that must not be confused with Enterprise Java Beans (EJB) that are part of the Java EE (Java Enterprise Edition) standard formerly known as J2EE, which addresses enterprise computing. DCOM (Distributed Component Object Model) is an example of a component technology that addresses distributed object computing. CORBA (Common Object Request Broker Architecture) [264] also supports distributed object computing, however, it is usually not mentioned as a component technology in its own right, only together with CCM (CORBA Component Model) [266] it is perceived as a component technology that addresses enterprise computing and is similar to Java EE. OSGi (Open Services Gateway Initiative) is an example of a Java-based component technology that enables systematic hot deployment of software , i.e., support for dynamic – in particular also remote – deployment and update. It is initiated and exploited by the embedded software community. A prominent usage of OSGi is as the foundation of the integrated development environment Eclipse.

In the domain of enterprise computing, object-oriented application server technologies like Java EE are perceived as component technologies. Here, component technology is seen as an extension of standard object-oriented language platforms with features for persistence – most typically in the form of support for object-relational mapping – distributed programming and transactional processing. For example, in [260] we find this view on component technology as an extension of object-oriented programming with features for concurrency, persistence and distribution – among possibly others. Classical transaction monitors [29, 138, 141] like IBM's CICS (Customer Information Control System) [61] or BEA's Tuxedo (Transactions for Unix, Extended for Distributed Operations) also offer such features and even beyond – they usually tightly integrate support for user interface programming and dialogue control.

An interesting example for a component technology in the domain of enterprise computing is IBM's San Francisco framework [39]. This framework is interesting because it is a rare example of an initiative that actively incorporated the sub industry aspect into its efforts from the beginning. The San Francisco framework is Java-based. The architecture of the framework consist of three layers [38], see Fig. 3.1, i.e., the foundation layer, the common business objects layer (CBO) and the core business processes layer. Independent software vendors can construct their solutions – typically for customers in a vertical domain – by customizing and reusing software entities from each of these three layers. The foundation layer [304] deals with the typical crosscutting problems of the enterprise computing domain, which are, basically, transactions, persistence and security but also others like national language support (NLS). Furthermore, the necessary support for distributed object computing is provided, i.e., an object!request broker (ORB), support for externalization and so on. With respect to distributed object computing the foundation layer was designed after the OMG standards CORBA and COS (CORBA Service) – though no formal adherence to these standards was targeted.

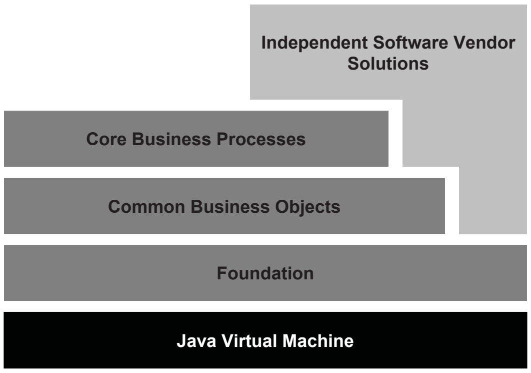

Fig. 3.1. System architecture of IBM's San Francisco framework.

With the common business objects layer the San Francisco framework starts to go beyond the discussed infrastructure aspect of component technology. The software entities provided here contain real business information and logic as default behavior. The software entities in this layer are rather general in the sense that they occur in several vertical domains. Examples for these entities are address, business partner, customer, calendar, time and currency [39, 193]. Actually also some design patterns [62, 130] for reuse in the next layer are implemented in the common business object layer. The next and highest layer of the framework, i.e., the core business processes layer, is about vertical domains. The software entities in this layer have been designed with domain experts from several companies in the particular domains. Examples for vertical domains addressed by this layer are the domain of business financials with support for, e.g., payable accounts, receivable accounts, and general ledger, the domain of order management with support for, e.g., sales and purchase orders, and the domain of warehouse management with support for, e.g., receiving and shipping of materials [193].

3.3.3 Large System Construction Aspect of Component Technology

Considering all the discussions on component technology we followed in the past we think it is fair to say that another important aspect of component-based development is simply that it is about the construction of large systems. It is common sense among developers that programming large system is fundamentally different from programming small systems. The larger a system becomes the more complex it becomes and you need special mechanisms to deal with the complexity. All the abstraction mechanisms in programming languages have the purpose to get complexity under control. The usual abstraction mechanisms found in programming languages are sufficient to build arbitrary layers of abstraction, so, in principle they are sufficient to deal with programs of any size. On the other hand, also small programs in the sense of programming in the large can be large enough for requiring the usage of programming language abstraction mechanisms in order to get into control of their complexity. So, the question arises: why do we need to discuss mechanisms that go beyond the usual programming language abstraction mechanisms? Or to pose the question differently: when is a program large, i.e., large in the sense of programming in the large [70]? One possible answer could be: programs are not large, projects [45] are.

In principle, each software system can be programmed by a single developer; however, often a wanted software system cannot be programmed by a single developer in a set time frame. Now, projects with more than one person differ fundamentally form single-person projects. There is overhead for communication, need for system documentation, need for system integration, and need for project management. Projects with more than one person, i.e., team projects are large. And programs that are developed in large projects are large. By the way, projects with distributed teams, i.e., sub-contractors, are usually even larger – that's why they are called mega projects in [135], a paper on the Boeing 777 software. So, team projects cost extra resources. And programming in the large actually addresses software programmed with more than one person. In the original paper [70] on programming in the large, or to be more precisely, in the paper that coined the term programming in the large, the notion of a module interconnection language (MIL) is introduced that should support developers in programming in the large. Two of the general objectives of the envisioned module interconnection languages explicitly address support for dealing with the overhead of division of labor. It is said that a module interconnection language should serve as a project management tool and as a means of communication between programming team members. Other objectives of the envisioned module interconnection languages are to serve as a tool for designing and concisely documenting large-scale program structures.

In answering our above question on when is a program large we said that one possible answer could be that programs are not large, but projects. We deliberately did not say that the answer is that programs are not large, but projects. Whenever a programmer feels overstrained with dealing with the complexity of a program he would be tempted to call the program a large program. This is a fuzzy characterization because defining when a developer is overstrained is not as easy as defining when a project is large – see above, we said that a project is large if it consists of more than one person. It would be superfluous to seek an answer to that question but the fact is that since the existence of programming languages we have seen a plethora of tools emerging that help programmers to get control of their code, e.g., profilers, shape analysis tools, style checkers, documentation generators, refactoring tools, versioning tools [84] to name a few and, last but not least, integrated development environments.

The major value added by an integrated development environment (IDE) is not that it combines several of the aforementioned tools as features but that it allows the developer to experience the code as a structure of hyperlinked code entities. For example, consider the major motivation of the aspect-oriented programming paradigm, i.e., the problem of maintaining the call positions of a code entity. This problem is also addressed in integrated development environments. Here, you can list the call positions of a method and the integrated development environment supports you in uniformly manipulating these call positions by its in-built refactoring capabilities. As any of these mechanisms that go beyond standard programming language features, also component technologies add value to the task of controlling complexity of large programs. Even if in academia the discussion of component technology is sometimes rather focused onto programming language constructs, in practice, a concrete component technology typically consists of a combination of new programming language features, tools, and software services.

Actually, in [45] yet another characterization of programs becoming large is given. There an artifact named "programming system product" is considered. A program which is implemented by a single person becomes a programming system if it consists of parts implemented by different programmers. In another dimension a program becomes a programming product if it is developed for more than one usage context. Different objectives like adaptivity, reusability and maintainability now become an issue. Brooks coined the term productizing for the transition from a program to a programming product – see Fig. 3.2, which also gives estimates for the extra efforts needed for the transitions in both dimension. With the consideration of productizing the loop is closed to the sub industry aspect of component technology that we have discussed earlier.

3.4 Exploiting Emerging Tools for BCM

We have discussed the importance of business continuity management for enterprises, how it targets the stability of an enterprise's business processes

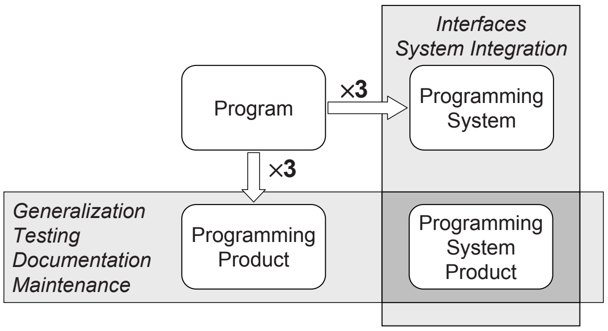

Fig. 3.2. Efforts for division of labour and productizing according to Frederik Brooks [45].

and the basic principles how business continuity can be achieved in Sect. 2.5. In order to support business continuity management there exists a range of proprietary tools, e.g., for writing business continuity plans, assessing risks, analyzing business impacts [32]. These tools usually come as combined structured editors, database repositories and bunches of templates for plans and questionnaires.

Both developing and eventually enforcing business continuity plans with the accompanying activities of risk and impact analysis are usually highly collaborative efforts if done properly. There are elements of knowledge management in these tasks and social aspects must not be neglected. Therefore, it seems natural to use some kind of CSCW tool (computer-supported collaborative work) or groupware [143] to get these things done. The several team collaboration software and social software products [116] that are currently emerging in the realm of the Web 2.0 metaphor form today's generation of CSCW tools.

According to [116] a social software product is expected to provide at least shared workspaces, management of shared documents, discussion forums, user profiles – all this supported by appropriate user and access control management – to count as a team collaboration and social software. Team collaboration software helps exchanging knowledge and joint building of knowledge bases. Therefore wikis [240, 215] and web logs (blogs) naturally fall into this software product category. Other features that fall into the area of team collaboration and social software, i.e., features that can be found in concrete products of this software category are about allocating and tracking tasks, managing projects, integration of calendars, controlling workflows, social tagging and bookmarking, visualization and analysis of social networks, content feeds, people search capabilities, in particular with respect to skill management, decision support for teams like support for prioritizing items, voting and ranking. Support for basic groupware features, i.e., email and team-based calendaring, or tight integration with respective products is expected. Also other traditional but more advanced groupware features like instant messaging and video conferencing belong to the repertoire of team collaboration software.

Against the background of all these just mentioned team-supporting software features team collaboration software seems a natural candidate as a tool in business continuity management. Imagine the process of risk assessment, impact analysis and finally estimation of risk probabilities. The processes for gathering the necessary information and afterwards categorizing and ranking the information items can be greatly supported by the features found in today's team collaboration software products.

Unfortunately, the unconsidered idea to support business continuity management by web-based team collaboration is naive against the background of the especially strict security needs of the considered domain. An important threat considered in business continuity management is always any kind of intrusion leading to several security actions from facility security to all the issues of IT security. For potential intruders the plans that deal with any kind of threats can be of interest. Therefore security requirements of the business continuity management domain are significantly high. People working in this domain and conducting the business continuity management often stem from the security sector or IT security sector. The problem is that people from the security sector are often biased against web-based technology; they often tend to work only with tools they have long experience with. Therefore, it can be challenging to convince stakeholders in the business continuity management process to use a web-based team collaboration platform. The openness of such platforms can be easily considered just too insecure. Of course, there are possibilities to make the usage of a web-based platform secure. The platform itself can be secured with virtual privacy network technology, but more obviously it is possible to fully separate a small intranet from the outside world and make it the basis for the team collaboration platform.

On the other hand, first experience already tells us that the capabilities of team collaboration and social software are really promising for business continuity management. At least we know this from one of our own projects where a simple wiki has been used in order to grasp and communicate the business continuity plan. Here, the wiki has proven particularly practical because it immediately integrates the business continuity plan with other existing documentation of the system architecture in a lightweight manner. A simple plain wiki system has been used. Mature domain-specific team collaboration software would extend the plain wiki software with templates and predefined workflows for business continuity planning and implementation.

3.5 Integration of Business and Production Processes

There is a huge potential for optimization of processes in today's industrial manufacturing. Important targets of improvement are production efficiency and product quality. Optimization is a complex task. A plethora of data that stems from numerical control and monitoring systems must be accessed, correlations in the information must be recognized, and rules that lead to improvement must be identified. Despite concrete standardization efforts existing approaches to this problem are often low-level and proprietary in today's manufacturing projects. The various manufacturing applications in a company must be turned from isolated spots of information into well-thought out integrated data sources [47, 16] that make up an overall solution.

The lowest level considered in automatic manufacturing is the automation level, i.e., the level of machine and device control. However, the automation level is not merely about automated tasks. For example, machine maintenance, transportation control and stock control are important issues at this level. The entities controlled by control computers are machines, cranes, transport belt systems or other transport mechanisms, chemical processors, converters etc. This is the level of computer numerical control (CNC), robot control (ROC), motion control (MC), programmable logic controllers (PLC), cell controllers (CC), data collection systems (DCS) and so on.

The technical integration of production devices is an issue in its own right. This is the domain of fieldbus technology like Modbus, CAN (Controller Area Network), PROFIBUS (Process Field Bus), AS-i (Aktuator Sensor Interface) – to name a few. Fielbusses are network protocols that have their strength usually at OSI level 2 – data link layer. A technology that addresses the issue of vertical integration of production devices immediately at the level of application programming interfaces is OPC ('Openness Productivity Collaboration' formerly known as 'OLE for process control') [269]. The technical integration of production devices is in a sense a horizontal integration. In the discussion of this section we are rather interested in the vertical integration of automation control shop floor control and production planning. However, vertical and horizontal integration are not completely orthogonal issues. In particular, a strictly data-centric horizontal integration approach could greatly ease vertical integration efforts from the outset.

3.5.1 Automatic Shop Floor Control

In a fully automatic plant in today's manufacturing industry there are automatic shop floor control systems or process control systems that control and track the interplay of the machines. This is the level that is often called SCADA (Supervisory Control and Data Acquisition). To give an impression we describe a fictional shop floor control system. Though the example system is fictional its described functionality is very close to a real world system from the area of material refinement. However, we abstract from the concrete domain and from the full complexity of the system, because the terminology and the details of the concrete domain do not add to the understanding of the concepts implemented by the software. In this example there is a fully integrated software control of all processors and transport devices in the plant. In a control center a supervisor sees a screen similar to the one depicted in

Fig. 3.3. Material is shipped into the plant in batches. The batches ship in from another plant in the factory. Each batch has to be processed sequentially by three kinds of converters A, B and C. There are several processors for each kind of processor, because the processing stages take different amount of time. For example, processing the batch by one of the B-processors takes approximately twice as much time as processing the batch by an A-processor. Therefore, there are twice as many B-processors as A-processors, otherwise, the B-processor stage would become a bottleneck.

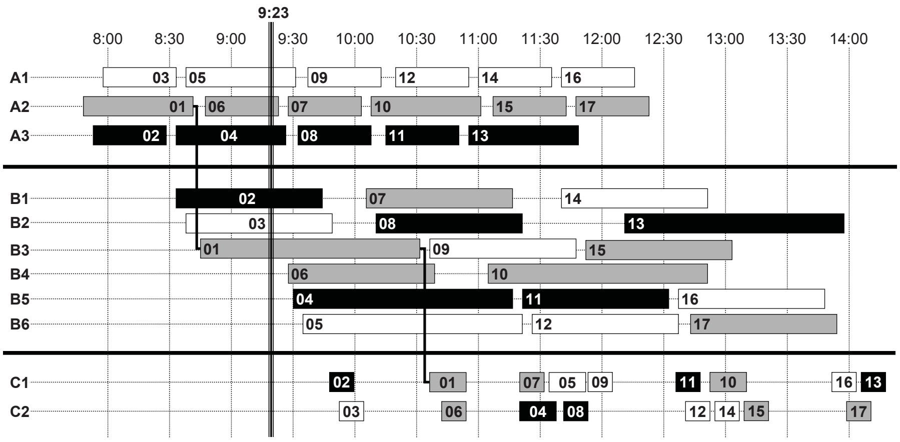

Fig. 3.3. An example manufacturing execution system.

In the graphical user interface in Fig. 3.3 the material flow proceeds from left to the right, from the top to the bottom. Actually, the graphical user interface shows a Gantt diagram of what is going on in the plant. A bar in the diagram stands for the processing of a batch in a certain processor. There are no edges connecting the bars in the diagram. The material flow is given by the numbering of the batches that remains the same throughout the different processing stages. Just to give an example, we have painted the edges for the material flow of batch with the number '01' into the diagram. A fat vertical line on the screen serves as the current-time indicator. The crucial point is that the diagram on the screen is not just about planning, it is really about control. If the current-time indicator passes the right end of a bar element, the corresponding batch is fully automatically removed from the current processor and moved into the next processor according to the schedule in the diagram. Workers in the plant are triggered by events on the production process and not vice versa. Workers can be considered to be embedded into the production process, i.e., they do not control it.

Even the scheduling, i.e., the assignment of processors to batches is done automatically by the system. The optimization target is resource utilization.

However, the supervisor in the control panel has the opportunity to reschedule the batches, i.e., to overwrite the default schedule proposed by the system. Furthermore, he can adjust the processing time per batch and processor. This way, he can react to exceptional events based on his expert knowledge. For example, he knows that a certain batch can only be processed on a certain processor, because this processor has a certain feature that is needed in the concrete case. Or one of the processors needs to pause for a while, for example, for maintenance reasons. Or the supervisor recognizes that a certain batch actually needs more processing time on its current processor than initially assumed. With respect to this it is necessary to know that the supervisor has a second screen on his desk which shows a dashboard with miscellaneous information about the current state of each of the processors.

3.5.2 Manufacturing Execution Systems

At the level of enterprise resource planning systems managers use production planning systems (PPS) for rough planning of the production. Rough planning means that managers use aggregate values for capacities and performance of production resources. Rough planning also means long-term planning, i.e., the time units managers deal with during production planning are rather months, weeks or days at the least. The management needs to give the planned production schedule to the production department as an internal order and needs feedback about the actual production in order to compare production figures with planning figures and to have a hook for high-level quality control and potential production process optimization. It also needs the feedback to improve its production planning process by an adjustment of the aggregate values used during planning.

Without further IT support there is a huge gap between production planning systems and the automation level. It is the task of manufacturing execution systems (MES) [202] to bridge this gap – see Fig. 3.4. Most importantly, with manufacturing execution systems production process planners detail the rough planning they receive from production planning to a level of detail at which shop control becomes possible. The time frames manufacturing execution systems deal with are much smaller than the ones of production planning systems – they are in the range of days, working shifts or even minutes. Full-automatic shop control systems allow for real-time planning, control and monitoring of a plant. Therefore manufacturing execution systems are natural hosts for shop control systems.

To give an impression of what manufacturing execution systems are about we list their functions as defined by the industrial standardization body MESA (Manufacturing Enterprise Solutions Association) [244]:

- Resource allocation and status.
- Operations and detail scheduling.
- Dispatching production units.

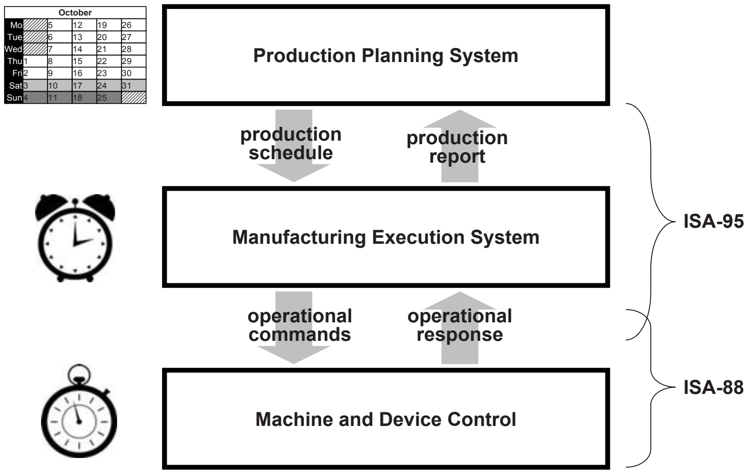

Fig. 3.4. Production planning, execution and control system architecture.

- Document control.
- Data collection acquisition.
- Labor management.
- Quality management.
- Process management.
- Maintenance management.
- Product tracking and genealogy.
- Performance analysis.

A manufacturing execution system supports the systematic fulfillment of the production schedule given by the management and also supports the delivery of the production reports needed by the management. It is that integration aspect between automation level and enterprise resource planning of manufacturing execution systems that is often emphasized. However, from the above list it becomes clear that a manufacturing execution system already adds significant value in the production department even if it were not connected with production planning systems. It supports daily operations with concrete features – maintenance management and quality management are good examples. On the other hand, it becomes also clear that a manufacturing system should be connected to the enterprise resource planning somehow; for example, consider labor management and human resources management.

3.5.3 Current Automation and Business IT Initiatives

We see better and better integration of production systems with enterprise resource planning systems as the current trend in information technology in manufacturing enterprises. It is fair to characterize this issue also as targeted integration of production processes and business processes. Actually, it is a bit odd, because from a conceptual viewpoint production processes are no different from business processes, on the contrary, they are business processes. However, it is common to use the term business process rather for administrative business processes, i.e., such processes that deal with enterprise resource planning, and therefore to distinguish them from the technical production processes.

In the following we use also the term automation and business integration for the integration of production systems with enterprise resource planning systems and even beyond with the integration of business intelligence (BI) systems.

Current initiatives like MESA and ISA (Instrumentation, Systems and Automation Society) address this vertical system integration issue. STEP (Standard of Product Model Data) [180] standardizes the description of both physical and functional aspects of products. ISO 15531 (MANDATE) [184, 66] provides a conceptual data model for manufacturing resources, manufacturing engineering data, and manufacturing control data. Both STEP and ISO 15531 are examples of standards that pursue a data-oriented viewpoint on the manufacturing scenario. As depicted in Fig. 3.4 ISA [314] addresses the standardization of models and terminology of batch processing at the automation level with ISA-88 [170] and the standardization of the information exchange between manufacturing execution and enterprise resource systems with ISA-95 [171, 172], see also [179]. In ISA-95 uses further terminology, in particular it uses 'Manufacturing Operations & Control' for the level of manufacturing execution systems and 'Business Planning & Logistics' for the level of enterprise resource planning.

The current trend of production and business process integration again has the objective to eventually lead to more overall flexibility and reactiveness of the enterprise. The features of a manufacturing execution system add benefit even if they do not lead to a measurable impact on reactiveness of the manufacturing enterprise. A manufacturing execution system can improve performance by speeding up the information flow between management and production and by optimizing the utilization of resources. By its data acquisition and reporting capabilities it can help to improve product quality. Anyhow, a foreseen improved reactiveness of the manufacturing enterprise is a major driving force for better integrated manufacturing execution systems. Major vendors make this argument in their current manufacturing IT initiatives like SAP with adaptive manufacturing [310]. At the technological level SAP's adaptive manufacturing initiative stands for the standardization of interfaces for third-party software vendors to SAP's own enterprise resource planning systems. At the strategic level SAP's adaptive manufacturing argues with an envisioned adaptivity of the manufacturing enterprise.

3.5.4 Industrial Information Integration Backbone

Current initiatives for automation and business integration take the situation of separate automation systems, manufacturing execution systems and enterprise resource planning systems as given and concentrate onto the clarification of the roles and responsibilities of these systems and the interfaces between these systems. This is a classical way of proceeding that we usually see in system integration trends. Certain classes of systems evolve and manifest themselves, then integration is about easing and standardizing the information flows between these systems. It is always worth considering more radical integration that creates a new class of system from scratch that unifies the systems that need to be integrated. In the case of automation and business integration we think it is interesting to think about the design of combined manufacturing execution and enterprise resource planning systems this way accomplishing integration from the outset. In such a system the different functionalities can remain software modules or software layers, however, they are integrated via a shared data model and database for which we coin the term industrial information integration backbone (IIIB) – see Fig. 3.5.

Arguments for Separation of Automation and Business Systems

There are also reasons to stick with the currently architecture of separate systems for manufacturing and enterprise resource planning. These are the usual reasons. One is the protection of investment with respect to exiting systems. Another is a make-or-buy decision in favor of buying available products and integrating them instead of building the whole system from scratch. Both enterprise resource planning systems and manufacturing execution systems are complex. Already for each class of system it must be carefully analyzed in a given scenario whether it is cheaper in the sense of total cost of ownership – see Sect. 2.6.3 – to deploy an existing commercial-off-the-shelf software system or to build an entirely new one. And these systems are so complex that there are specialized vendors for each of them – we already mentioned SAP's adaptive manufacturing approach to integrate manufacturing execution systems by third party vendors. Another reason is the desire to address different levels of quality of service of different applications with appropriate organizational structures along distributed application servers. For example, consider availability. Enterprise resource planning systems might not require high availability in an enterprise, whereas the availability of manufacturing execution systems – at least the availability of shop floor control systems – is easily a mission-critical issue. Similarly, there is often what we call an ownership issue or self-sustainment issue, i.e., the fact that different IT systems are built and maintained along the organizational structure of an enterprise driven by departments that sometimes long for as much independency from other business units as possible.

A further counter argument is simply that the functionalities of a manufacturing execution and enterprise resource planning system is simply too extensive to be delivered by a single vendor. In particular, the argument is that there must be specialization of the systems to meet different needs of different enterprises that stem, e.g., from vertical domains or the concrete sizes of the enterprises. Obviously, in the indicated field there must be doubt that it is possible to build a system that can fulfill the needs of all the diverse manufacturers – the no one-fits-all problem. All this is a sub industry argument. At least it could be the argument that because of the large amount of functionality it is desired to build the optimal solution in a concrete scenario by combining it from different software vendors. With respect to this counter argument it is interesting to see that ORACLE outlined in [309] its general product direction for manufacturing execution systems towards a single combined ERP/MES application. Actually, in April 2007 Oracle released the Oracle Manufacturing Execution System as part of its E-Business Suite, which is an enterprise resource planning system. The solution has been announced as a product for enterprises that operate in environments of low to medium complexity.

Arguments for Integration of Automation and Business Systems

All of the above are counter arguments against the concept of an industrial information integration backbone approach. However, there is a single but very strong argument for the architecture to integrate manufacturing and business IT via the database from the outset and this is flexibility. It is just the principle of data independency, i.e., the principle of centrally designing, operating and maintaining the data independent from the applications that exploit them, that improves flexibility of the total information system.

In general, having a database as a central hub for integration is a proven pattern as is already inherent in the currently widely discussed service-oriented architecture and explicitly seen in enterprise service bus technology. In the original enterprise computing related strand of service-oriented architecture – see Sect. 8.2 for a discussion and Fig. 8.2 in particular – the services in service-oriented architecture form a hub in a hub-and-spoke architecture of applications that this way are integrated and use each other in a flexible manner. It is not essential that the services tier in a service-oriented architecture possesses its own database, i.e., in principle the service tier can be a mere message generator collecting data from the applications it integrates on the fly and distributing them. However, it is a typical technical pattern that the service tier has its own database to persistently buffer data. This is where the service tier begins to become an enterprise service bus which is also discussed in Sect. 8.4. The notion of enterprise service bus is a loose concept for enterprise application integration that combines persistent messaging, in particular, publisher-subscribe functionality, with new features like content-based routing in the realm of web services technology, i.e., enterprise service busses are the web-services related instances of message-oriented middleware. And indeed, established persistent messaging technologies like IBM MQSeries [350] are in their own right examples of technology for integration of applications via a database. However, it is fair to say that the driving force for the exploitation of persistent messaging was not step-wise enterprise application integration but building lightweight but at the same time still robust alternatives to distributed transactions in transaction processing systems that are distributed on a geographical scale.

The counter arguments against the integration backbone discussed earlier are pragmatic reasons that pay tribute to existing system architectures that evolved. The standardization of message flow between applications makes the market for these applications more agile by bringing flexibility into the decision-making of customers in selecting a concrete product, but it does not address the flexibility of the systems themselves fundamentally.

Like any other approach to design a unified automation and business IT product the industrial information integration backbone does not address the aforementioned no one-fits-all problem. However, the integration backbone is an architectural principle. Not all systems are bought because of careful build-or-buy decisions. So, if a system is built, for example, for an enterprise in a special vertical domain, the integration backbone can be a design option. We just say that in such cases the design efforts should not be automatically directed and possibly misled by the existing and emerging industrial integration standards, because those standards arose to improve the message flows between applications in de-facto scenarios of manufacturing enterprises. They should not be taken without review of blueprints for building a system from scratch.

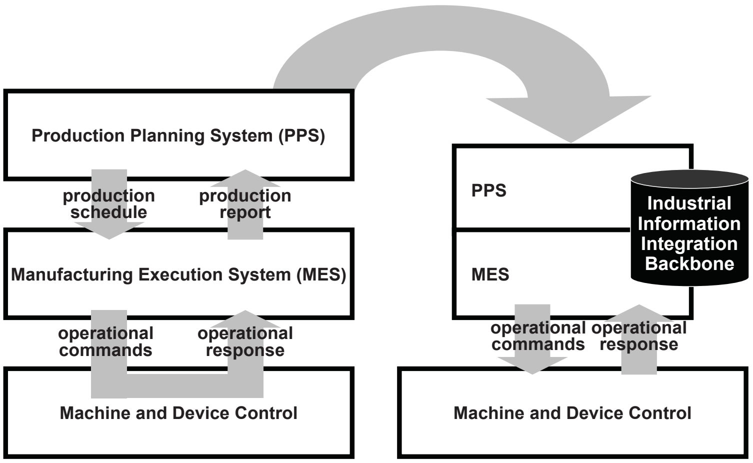
Production Planning System (PPS)
production schedule
production report
Manufacturing Execution System (MES)
operational commands
operational response
Machine and Device Control
PPS
MES
Industrial Information Integration Backbone
operational commands
operational response
Machine and Device Control

Fig. 3.5. Industrial information integration backbone.

The discussion of whether using a message-based approach or a data-based approach to the integration of manufacturing execution systems and production planning systems is an instance of a general discussion of distributed versus centralized systems. The aggregated driving force of such discussions has to be total cost of ownership – see Sect. 2.6.3. Concrete typical driving forces in such general discussion can be price, performance and reliability [140]. However, it is not clear from the outset which architecture is cost optimal in a concrete situation. For example, the robustness of a distributed system built from low cost components can be better than the robustness of a centralized system built from high cost components [228]. As long as the community lacks a constructive cost model on the basis of standardized software system components, software architecture will remain heuristics-based.

3.6 Integration of Business Processes and Business Intelligence

So far, we have discussed in Sect. 3.5 the integration of manufacturing execution systems and enterprise resource planning systems. A similar architectural discussion arises when looking at the integration of enterprise resource planning and business intelligence. Beyond the already state-of-the art point-of-sales analyses their is an ongoing trend in systematic business activity monitoring (BAM) [117, 236]. The standard architecture enabling analytical processing in today's enterprises has separated online transaction processing (OLTP) and online analytical processing (OLAP) systems. These systems are really separated, i.e., they consist of software that resides on different servers. Between these systems there is yet another system, often also on a separate server, that is responsible for the extraction of data from the transaction system, the transformation of the data into formats that are suitable for analytical processing and the transportation – called load – of the data into the analytical system. This latter man-in-the-middle system is called ETL layer (extraction, transformation, load). Conceptually the point is that there are two kinds of system, i.e., systems that are there for daily operations in the enterprise and systems that are there for analyzing the outcome of these daily operations and – with the current trend of business activity monitoring – also for observing the daily operations themselves. In practice, you can find also systems that combine the three layers onto one server, but these systems are then ad-hoc solutions in small, uncomplex business environments. If the layers are run on a single server in practice this is not about creating an innovative data warehousing architecture like the one we are discussing in the sequel but just about exploiting available business intelligence products on a simple server infrastructure wherever possible for occasional analyses. However, what we are talking about here is systematic analytical processing on a large scale, so ad-hoc architectural alternatives are not of interest here.

In the discussion of data warehousing architecture the term analytical processing fits the intention of these systems, whereas, at a first sight, the term transaction processing may seem to be a bit odd for the systems that support daily operations. It seems odd, because one connects the term transaction with technical concepts like ACID transactions or transaction monitors. Even in this sense, the usage of the word transaction processing system for systems that support daily operations is a good fit, because it is correct that the technical notion of transaction is dominant in these systems. Anyhow, the term transaction processing is quite good, because it can be understood as hinting not to technical concepts but the ephemeral nature of data emerging and disappearing in IT systems that support daily operations – we will have closer look onto this topic in due course.

Again it is compelling – both from a scientific viewpoint but also from an innovative product viewpoint – to think about a radically different system architecture that integrates the systems under consideration from scratch. In such architecture the schemas that form the basis for transactional processing – called transactional schemas for short in the following – and the schemas that form the basis for analytical processing – similarly called analytical schemas – reside in the same integrating database. In such an architecture the analytical schemas are views on the transactional schemas and the definitions of the view update mechanisms correspond to the ETL layer of current data warehousing (DW) architectures. The architectural notion of integrating via the database in this case pays tribute to the increasing hunger for more and more data extraction and shorter and shorter update cycles for the analytical data [44]. The significantly shortening of the extraction and transformation times in concrete data warehousing architectures is one of two aspects of the current active data warehousing (ADW) trend, which is an issue both in industry [149] and academia [257]. The other aspect of active data warehousing is about closing [343] the loop between analytical and transactional systems, i.e., feeding back information from analysis to operations automatically and exploiting analytical data in rules that control business logic and business processes in IT for daily operations. The closed loop aspect of active data warehousing is another argument to consider the integration backbone approach. Together with the foreseen need to exchange information between transactional and analytical systems eventually in real-time it actually leads somehow naturally to this approach.

3.6.1 The Origin of Today's Data Warehousing Architecture

As with most ubiquitous system architectures there are two kinds of reason why data warehousing architectures today look the way they are. The first kind of reasons is about how the systems emerged; the second kind of reasons has to do with concrete pragmatic issues of system operations. Both kinds of reasons are mutually dependent. With respect to the first, i.e., the evolution of today's data warehousing architecture it has to be understood first that there have always been different kinds of let us say functionality groups of enterprise information systems and different kinds of data. Let us approach this by taking a data-centric viewpoint.

Classically, it is usual to distinguish between master data, transaction data and inventory data. Today there is actually one more kind of data, i.e., analytical data and that is the point as we will discuss later. As you will see, the distinction between the classes of data is fuzzy and with time the boarders between them diminishes more and more. The master data of an enterprise are those data that must be available for usage by many applications over a long period of time. They are updated seldom and therefore they are also sometimes called fixed data or basic data. Typical examples of master data are customer data, supplier data, article data or personnel data. Transaction data are permanently new arising data. They are captured during the execution of daily business processes. Examples of transaction data are accounting transactions, reservations, purchase orders, bills, receipts. The life time of transaction data is limited from the outset. However, often you can find them consolidated and aggregated as analytical data in data warehouses. Transaction data are exploited in that they impact the update of inventory data. The inventory data represent the business figures; they originate from the accumulation of transaction data. Like master data, inventory data are stored for a long period of time. Typical examples of inventory data are account balances, business volumes and goods in stock. This means that the notion of inventory data is more comprehensive than inventory data in the narrow sense, i.e., data about goods in stock – it is accumulated data about all the goods and values in the enterprise. In some commercial-off-the-shelf enterprise applications you will find a more coarse-grained distinction between master data and transaction data only – the inventory data are then usually subsumed under master data. However, it is actually the existence of inventory data in transaction processing systems that interests us here, because its consideration is very instructive in the discussion of data warehousing architecture.

The question is why the different kind of data are distinguished by their typical duration. Why not just store all transaction data forever? One answer lies in technical limitations. Storing all transaction data means maintaining a log of the enterprise life stream [101] and this is just too much data to be stored. However, with more and more computing server power available – see the results of the benchmarks by the Transaction Processing Council (TPC), e.g. [345, 346], and Storage Performance Council (SPC) – the argument becomes weaker. For example, for years the retailer Wal-Mart stores data about each shopping cart, i.e., sales figures and data about the products sold within one customer transaction, in its data warehouse [355] — have a first look at Fig. 3.6 – resulting into a data volume of 600 tera bytes in 2006 [14]. Other reasons for not storing all transaction data can be found in the topic of data protection and here, in particular, in a need for adherence to law regulations. Actually, with respect to this issue also the converse is true. Currently, we see a trend towards systematic business transparency – think of Sarbanes-Oxley

Act (SOX) [34] and Basel II. A lot of enterprise are currently challenged with implementing crosscutting data auditing [258, 214] mechanisms that ideally record all data messages exchanged in the enterprise IT for later analysis.

Along the lines of the different kind of data just discussed there have always been different kinds of functionality in enterprise applications. The first one is about gathering data from daily operations and processing them. The reports that are generated in this operations mode are usually lightweight and they serve only to enable daily operations and transactions. The reports are not there for business analysis. The second kind of functionality is about generating complex reports on the basis of inventory data. These reports are needed in controlling and planning. This means that reports for decision support could always be found in enterprise applications. Over the time the potential for systematic multi-dimensional transformation of the transaction data for supporting decision support has been recognized. Also data mining with its algorithms to discover correlations and dependencies between stochastic variables entered the scene. Dedicated decision support systems were built, data marts that deal with chunks of enterprise data to address particular required analyses and also holistic data warehouses.

It is a usual phenomenon that enterprise IT systems grow to system landscapes, because new needed functionality are not introduced as new features of existing applications but introduced as additional software and server systems. So is in the area of analytical data. With the need for a new generation of analytical capabilities new supplementing decision support systems were introduced. The organizational pattern behind this is to never touch a running system. Analytical processing is technically cost intensive, i.e., it longs for significant extra server computing power. In the aforementioned analytical processing example of Wal-Mart a 1000 node massively parallel computer by NCR/Teradata was used in 2006 to deal with the 600 tera byte analytical data. The Wal-Mart example is an extreme example of a high-end data warehousing solution – indeed, it is fair to say that Wal-Mart has been the outrider for data warehousing. However, the example gives an impression of the relative cost-intensiveness of analytical processing. Still it is not possible for small and medium enterprises to buy high-end data warehousing solutions. A new solution must allow for precise determining the risk of burdening an existing IT infrastructure, which is responsible for supporting daily operations with a robust quality of service, with extra load for cost-intensive analytical queries. Therefore, it is the correct architecture to build a separate system for this solution that is allowed to connect the existing system only occasionally and for limited durations, typically in times where the transactional system is known to be rather unused – during the night for example.

3.6.2 Marrying Transactional and Analytical Schemas

Unifying transactional schemas and analytical schemas into a single database server holds the potential for an unseen degree of connection between trans-

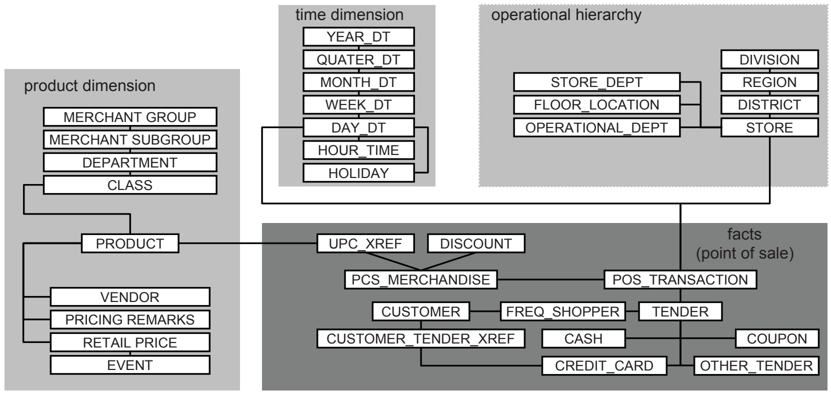

Fig. 3.6. Cut-out of the Wal-Mart data warehouse schema.

actional and analytical processing – with respect to speed, maintainability and possible utilization. This is not about throwing together existing products onto the same server. It is about making transactional and analytical schemas the unified basis for both transactional and analytical applications, again in the sense of data independency. Transactional schemas and analytical schemas are just regions of a whole database schema in such a solution. Programs that transform and transport data between these regions replace current ETL layers and these programs are just further applications that access and manipulate the data in the sense of data independency. Here, in principle, it does not matter whether these transforming programs reside on a separate application server or on the database server itself, however, it is very likely that they will be placed onto the database server exploiting active database features.

Application Separability

The problem with each combined transactional and analytical solution is that it must support what we call separability or application separability. Separability is the possibility to guarantee the robustness or quality of service of one application independent from the influence of other applications in a system of applications. For example, operating system processes are a concrete mechanism that target separability. In multi-tier system architectures separability becomes a subtle issue. The classical tiered data warehousing architecture naturally supports quite strong separability, because the systems are actually separated. The connection between the systems is limited to the times the ETL layer reads from the transactional systems. ETL layer products support the maintenance of this connection by providing means to schedule the extraction. For the separability of the envisioned architecture database management system features are necessary that allow for an advanced prioritization of the database tables and the threads accessing these tables along the lines of a mature access model. Actually, commercial database products offer such advanced features.

As a proof of concept, a first step in the direction of fully integrated transactional and analytical processing could be undertaken by running existing database, ETL layer and data warehousing products in separated capsules of an appropriate virtualization software or of an operating system that natively supports virtualization like i5/OS with its hypervisor. Such an approach yields separability, however, in the beginning there is obviously no advantage in terms of tighter interweaving of transactional and analytical processing. However, once the system is running on the same machine it can be seen as whole and it can be patched in a very targeted way to try out potential speed ups of the extraction and load processes. Technically, such an attempt is only possible if the system is a complete white box to the experimenting developers. So, natural candidates in such an attempt are open-source products for the data warehousing technology, e.g., Mondrian (Pentaho analysis services), for the database technology and, in particular, for the virtualization software, e.g., Xen [78].

Completely Crosscutting Information Backbone

Systems evolve. New systems with new functionality are added while the impact and value of these systems can not be really estimated at the time of their introduction. From time to time it can be fruitful to analyze for a certain kind of enterprise functionality whether the driving forces on the problem side and the existing architectural patterns on the solution side are fully understood. If some stage of maturity is reached it is time to think about systematically designing a unifying architecture from scratch.

In Sect.3.5 we have discussed the integration of manufacturing exececution systems with enterprise resource planning systems via the database, in this section we have discussed the integration of enterprise resource planning systems with analytical processing via the database. This eventually leads to an extension of the industrial integration backbone so that it spawns all the different kinds of applications discussed [114]. We have visualized the resulting architecture once more in Fig. 3.7.

Once such a database backbone is created, in particular, with the unification of transactional and analytical schemas, it would be possible to think about thoroughly applying multidimensional schema design over all levels of data in the functionality stack. Today, it is state-of-the art to exploit multi-dimensional schema design for point-of-sales analysis, see the Wal-Mart database schema in Fig. 3.6. It is possible to ask what has been sold, when, where, why, by whom, to whom, why? The analytical power lies in the opportunities to drill down and roll up the dimensions of this question. With respect to activity monitoring exploiting multidimensional schema design [236] means to pose similar question about who did what, when, why, with which resources, results, performance etc.? Imagine the analytical potential in combining this with the data from the production process, eventually leading to an IT system integration from the top floor to the shop floor.

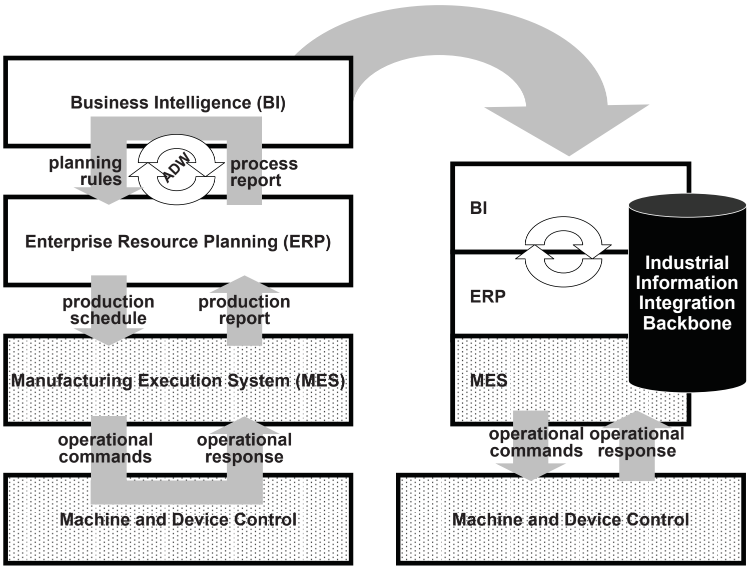
Business Intelligence (BI)
planning rules
ADW
process report
Enterprise Resource Planning (ERP)
production schedule
production report
Manufacturing Execution System (MES)
operational commands
operational response
Machine and Device Control
BI
ERP
MES
Industrial Information Integration Backbone
operational commands
operational response
Machine and Device Control

Fig. 3.7. Completely crosscutting information backbone.

Information Backbone Compared to Data Mart Architecture

In Sect. 3.5.4 we conducted a discussion about the arguments for the integration versus arguments for the separation of automation and business systems. This discussion was conducted against the background of enterprise resource planning systems and manufacturing systems. It is instructive to repeat this discussion here with the viewpoint of analytical processing of manufacturing data. The analytical processing systems in Fig. 3.7 form a business intelligence layer that is placed on top of the enterprise resource planning systems layer. Actually, there exists also analytical processing that directly gets its data from the manufacturing systems. Unlike business intelligence such analytical processing is typically not there for supporting strategic planning and decision support but for supervising and improving the production processes. Analyses like those found in six sigma projects – see Sect. 2.4.2 – would be typical. If such systems are also integrated via the information backbone approach this would result into an architecture depicted in Fig. 3.8. We have already mentioned in Sect. 3.5.4 that the information integration backbone architecture is a hub-and-spoke architecture and this aspect is visualized better in Fig. 3.8 than in Fig. 3.7.

Analytical processing in order to improve the quality of a production process means mining production data. It is very typical that the data necessary for a concrete data analysis are stored in a dedicated data mart. This way a landscape of data marts grows with one data mart for each kind of analysis. A practitioner's argument against a centralized database approach that replaces the data mart landscape is a performance argument. It is said that bringing back the analysis data into the production databases would unacceptably slow down this production database. An analysis is made of complex algorithms and queries against a data mart and a complex query that gather data from production databases. If it is possible to do several analyses on some extracted data it is reasonable with respect to performance to separate the extraction efforts from the analyses efforts. With a data mart landscape approach such a separation is enforced by system architecture. But the fact that a concrete system architecture enforces an architectural principle is not a strong argument for this system architecture. It is also possible to rebuild the data mart schemas in a central production database with the same performance benefits – off-loading in the shell so to speak. The data marts are then realized as non-updatable views, i.e., as materialized queries. It is often an option to save computing time by pre-computing results or manifesting part computations as reusable data – think of all the several kinds of indexes for information retrieval, for example.

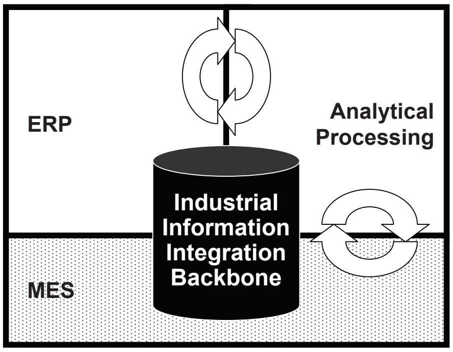

Fig. 3.8. Direct analytical processing for manufacturing data.

Nevertheless, it also has to be said that on the basis of current technology the integration approach really does not have to be the optimal one in terms of total cost of ownership. Bringing back data mart as a schema into the central database can lead to extra costs if the central database only scales up relatively expensive. If the data mart is, for example, relatively small and does not require the same service level agreement, it can easily be that the realization with an extra commodity server is the best price option. Furthermore, there might be provisos against integrating the data marts into the central production database by the production database owners. In a concrete case the dedicated data mart server solution is usually just the standard one in the sense that it effects the overall systems operations and is easier to estimate. Again, these counter-arguments against an information backbone approach must be traded off against its significantly improved conceptual maintainability and flexibility. It remains a research issue here to establish a system of quantifiable driving forces onto the selection of the costs and risks of the optimal information system architecture.