ELSEVIER Data & Knowledge Engineering 27 (1998) 31–57

DATA & KNOWLEDGE ENGINEERING

A reference architecture for workflow management systems

Paul Grefen[a,*], Remmert Remmerts de Vries[b]

[a]*University of Twenty, P.O. Box 217, 7500 AE Enschede, The Netherlands*
[b]*Bakkenist Management Consultants, The Netherlands*

Received 19 August 1997; revised manuscript received 28 October 1997; accepted 4 December 1997

Abstract

In the workflow management field, fast developments are taking place. A growing number of systems is currently under development, both in academic and commercial environments. Consequently, a wide variety of ad hoc architectures has come into existence. Reference models are necessary, however, to allow for the construction of cooperative systems, modular configuration of heterogeneous systems, and assessment of existing systems. Current standardization efforts offer limited help towards detailed software architectures. In this paper, we address this problem with a reference architecture for a full-fledged workflow management system, designed to comply with a broad set of aspects, such as heterogeneous environments, extensible functionality, and mobile workflow clients. The architecture goes into deeper detail and is more complete than existing proposals. It provides an elaborate software engineering framework for structuring systems under development and a functional framework for assessing existing systems. © 1998 Elsevier Science B.V.

Keywords: Workflow management; Workflow management system; Software architecture; Reference architecture; Complex system design

1. Introduction

In the field of workflow management systems (WFMSs), a very fast development is taking place. This development is driven by the general observation that workflow management can greatly increase effectiveness and efficiency of many administrative processes, both in commercial and public environments. The development of new workflow management systems is influenced by increase of both functional and technical requirements.

Functional requirements, as dictated by sophisticated workflow applications, are rapidly getting more diverse and complex. Workflow management systems must support complex procedures, often involving many actors, which may be geographically distributed. Workflow management systems should allow flexible task allocation to actors in the context of advanced organization models. Further, they should manage the scheduling of scarce organizational resources, they should handle a wide

* Corresponding author. E-mail: grefen@cs.utwente.nl

variety of exceptions in workflow execution, etc. Functional requirements vary per specific application area: application areas where meeting deadlines is an essential aspect put an emphasis on advanced scheduling facilities, whereas areas with frequent changes in personnel require flexible task allocation mechanisms. Further, functional requirements may be extended over time for a specific application. To cope with these aspects, a flexible system architecture is needed that allows for adaptation and extension. Thus, flexibility and extensibility are key issues for workflow management systems.

Technical requirements to workflow management systems are also rapidly getting more complex. An important factor is the high level of openness that is required to enable interoperability with a wide variety of other information systems, often of a legacy nature, and other workflow systems [3]. Also, platform independence is an important requirement to accommodate technically heterogeneous office environments. Further, scaleable performance is highly desirable to accommodate applications that grow in complexity and required throughput, and organizations that grow in size. As with functional requirements, technical requirements too vary per application area and may change over time for a specific application. Clear architectures are required to provide a stable basis for the above developments.

Many existing workflow management systems are developed from existing software systems, like document information systems, electronic mail systems and groupware systems. Consequently, a wide variety of ad hoc architectures has come into existence in which a common framework is hard to detect. Architecture descriptions of commercial systems are usually of a high level of abstraction and consequently of a low level of detail, if available at all. Architecture descriptions not related to commercial products usually have the same characteristics, see e.g. [38]. Overview papers on workflow management, e.g. [21,3], usually lack detailed discussions on architectures of workflow management systems.

A general, detailed architectural framework for workflow management systems is required, however, to be able to assess and compare systems and to provide a basis for interoperability between systems. Further, the 'ad hoc' architectures of existing systems limit the flexibility and extensibility of systems and do not allow for configuration of heterogeneous modular systems.

1.1. The Mercurius initiative

This paper presents a detailed reference architecture for an 'ideal' workflow management system developed in the context of the Mercurius initiative. Mercurius is a cooperation effort between two universities and a polytechnic, a management consultancy and software firm, and a large bank, aiming at the development of a reference specification for workflow management systems [32].

The system architecture developed in the initiative and presented in this paper is called 'ideal' for three reasons. Firstly, the architecture is not related to any existing workflow management system, which means that its design is not influenced by 'legacy decisions'. Secondly, the design of the architecture has been governed by an explicit set of design principles (see Section 3) to guard the quality of the design. Thirdly, the design has been performed in the context of a project team consisting of professionals from the research, consulting, and end user communities, thus ensuring completeness of the design. Finally, a broad range of functional aspects has been included in the architecture to obtain a complete framework.

The architecture presented in this paper has a high degree of modularity and flexibility, both with respect to functional and technical requirements. This is obtained by a multi-level modular architecture, the use of extension modules that can be hooked to a software bus, and the use of interface modules that shield platform-specific details from the core workflow management system. The proposed reference architecture can be used for various purposes. Firstly, it can be used as a benchmark for the comparison of architecture and functionality of existing workflow management systems. Secondly, it is a solid basis for the development of new workflow management systems. Thirdly, it is suitable as a framework for the configuration of modular architectures.

1.2. Related work

Recently, a growing number of initiatives in the field of workflow management has come into existence, both in the academic and commercial areas. The work focuses on many aspects of workflow management, like workflow paradigms and models, workflow design techniques, workflow specification languages, workflow support technologies, etc. (see e.g. [21] for an overview). Given this fast growing amount of work, it is notable that relatively little attention has been paid to detailed architectures for workflow management systems.

Currently, the Workflow Management Coalition (WfMC) is a well known initiative to develop standards for workflow management systems [43]. Part of these standards is the reference architecture depicted in Fig. 1. This architecture focuses on the different kinds of interfaces with the workflow enactment service, e.g. the interface with workflow clients and the interface with other enactment services. The standard does not include a detailed software architecture that can be used as a reference for the design of the internals of a workflow management system architecture. The relation between our work and the WfMC architecture will be further discussed in the sequel of this paper.

In the software industry, many workflow management systems are currently being developed. In [38], a general overview is given. Examples of commercial products are Action Workflow by Action Technologies [31], InConcert by X-Soft (Xerox) [30] and Flowmark by IBM [29]. Usually, the internal architectures of these commercial systems are not described in detail for the public. Further, architectures that are described in this context are usually to a high-level system-specific and therefore hard to compare or use in other contexts.

Several research projects are currently working on advanced architectures for workflow management systems. An example is the WIDE project, in which an architecture with orthogonal support for

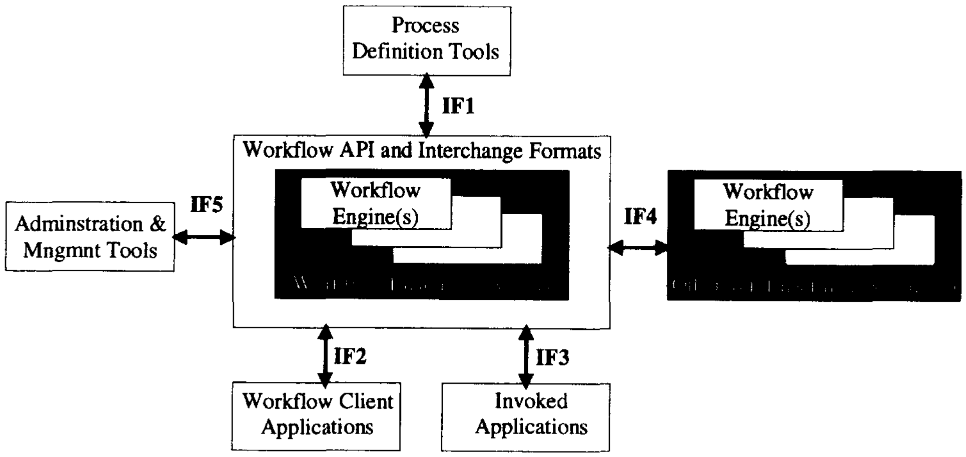

Fig. 1. WfMC standard architecture.

extended transaction management and active rule support is constructed [10,13]. WIDE focuses on the integration of specific extended database technology and workflow management technology, whereas the work presented in this paper focuses on general WFMS architectures. In the Exotica project, the Flowmark architecture is extended to deal with advanced features like advanced transaction emulation and distributed data management [1,2]. In contrast to the work in this paper, the focus in Exotica is on specific extensions to a specific WFMS platform. Another example in this category is the METEOR project, which focuses on support for multi-system workflow applications [28].

In contexts other than workflow management, reference architectures have been around quite some time. The best known examples are probably the ISO-OSI 7-layer network architecture [41] and the ANSI-SPARC three-schema architecture for database systems [42]. Another example is the OSCA architecture, which defines a layered architecture in the context of the telecom industry [7]. An example of a reference architecture specific to an application area is the data warehouse reference architecture published in [25].

In process-oriented fields other than workflow management, architectures have been designed to deal with process management. An example is the field of process-centered software engineering environments (see for example [18,37]). Where workflow management focuses on improving general administrative processes, software process support focuses on models and technology to improve processes in complex software production. An example in this context is the ALF project [9], in which an integrated software project support environment architecture has been designed that allows for definition and enactment of project-specific software production process support.

1.3. Structure of this paper

The structure of this paper follows the top-down design of the workflow management system architecture. Section 2 presents the global system architecture, in which the WFM system is embedded. The starting points for the development of the architecture of the WFM system are discussed in Section 3. In Section 4, the global WFM system architecture and the required workflow data sets are discussed. Section 5 describes the detailed system architecture for workflow design. The workflow enactment server architecture is elaborated in Section 6. The workflow client architecture is discussed in Section 7. Section 8 pays attention to the interfaces between the workflow management system and the software platforms it uses. The paper ends with some conclusions.

Modules of the reference architecture are illustrated by comparing them to modules of workflow management systems that currently exist or are under development, e.g. in the WIDE and Exotica projects.

2. Global system architecture

This section describes the overall system architecture in which the workflow management system (WFMS) is embedded, i.e. the WFMS and the various software platforms it uses. First, the system layers in this architecture are described. Next, attention is paid to the technical boundaries that can be distinguished in the overall architecture.

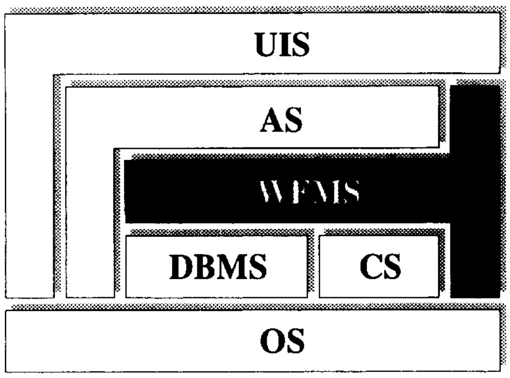

Fig. 2. Global system architecture.

2.1. System layers

The following software layers can generally be distinguished in a workflow management system software environment (see Fig. 2).

The *user interface system* (UIS) is used both by application systems and modules of the workflow management system (see Section 3) to implement the interactive interfaces to the users of the systems. This layer preferably offers a graphical window interface, but may also offer a text-based forms interface.

The *application systems* (AS) handle specific application data in the workflow processes, e.g. word processors, spreadsheet programs, data entry programs. Application systems can be interactive end-user applications or batch applications that perform operations in the background.

The *workflow management system* (WFMS) provides functionality for workflow design and workflow enactment (execution). This layer is the focus of this paper; it is further elaborated in the next sections.

The *database management system* (DBMS) is a multi-user store manager for all relevant workflow data sets (further discussed in Section 4.3). The database management system may be of a distributed nature.

The *communication system* (CS) provides communication services to other (peer) workflow management systems. It may be based on e.g. electronic mail or EDI.

Finally, the *operating system* (OS) layer provides basic system functionality to all other layers, such as file system functions. It may be heterogeneous in case of client/server architectures.

2.2. Intra-system boundaries

A number of important intra-system boundaries can be distinguished in the layered model of Fig. 2. These are discussed below.

In the first place, we can distinguish several transparency boundaries in the overall architecture. The distribution transparency boundary determines which system layers explicitly address distribution aspects and which layers can address the underlying layers as if it were a centralized system. In the architecture of Fig. 2, distribution aspects can be transparently hidden by the operating system layer

[14], by the database system layer [12], by the workflow management system layer, by the application system layer, or not at all. In the latter case, the end user of the workflow system is responsible for handling distribution aspects, which is usually not preferable. The platform transparency boundary determines which layer hides platform specific details from the higher layers. The lower in the architecture platform transparency is offered, the more general the higher layers can be. Note that there may be multiple platform transparency boundaries for specific platform-dependent aspects.

In the second place, we can distinguish one or more client/server interface boundaries [14]. A client/server interface can be used within the WFMS or between WFMS and AS. In the first case, WFMS clients are distinguished that provide the end user side of workflow management. In the second case, the end user side is implemented through external application systems. In the Mercurius architecture, both options can be used, although workflow clients are explicitly included in the description. Note that client/server issues are interesting only for workflow enactment aspects. In workflow application design, no central–decentral process coordination is necessary within the WFMS. Note further that a second client/server interface may be present between WFMS and DBMS, as most modern database systems for large applications have a client/server architecture.

3. Architecture design approach

In this section, we discuss the approach for the design of the WFM system reference architecture. First, we give our vision on the function and contents of a reference architecture for workflow management systems. Then, we discuss the various design principles that have been the 'foundation' for the design process. Last but not least, we pay attention to major architectural aspects that have heavily influenced the design of the reference architecture.

The use of the design principles and architectural aspects described in this section results in an open WFMS architecture with a high degree of flexibility, extensibility and portability. Flexibility and extensibility are obtained for both the WFM system developer and workflow application designer perspectives, i.e. for WFMS provider and WFMS user.

3.1. Reference architecture

In this paper we describe a *reference architecture* for workflow management systems. The architecture has two important characteristics that distinguish it from concrete WFMS architectures: abstraction and completeness.

The reference architecture presented in the sequel of this paper can be seen as an *abstraction* from specific WFMS architectures. This means that both low-level details and details specific to concrete systems are excluded from the reference architecture. The step from the abstract reference architecture to a concrete system architecture thus implies filling in system specific details at the appropriate places. We give examples of this in the sequel of this paper.

The reference architecture aims at *completeness* with respect to WFMS functionality. As such, it is functionally more complete than most (if not all) concrete systems. The reference architecture can be used as a functional 'benchmark' for concrete architectures. The step from the functionally complete reference architecture to a less complete concrete architecture implies selecting the appropriate modules from the reference architecture. In the sequel of this paper, we discuss which parts of the reference architecture are considered 'mandatory' and which parts 'optional'.

As stated before, the reference architecture discussed in this paper is not based on an existing workflow management system. To obtain the desired level of abstraction and completeness as mentioned above on the one hand and avoid 'legacy design issues', we have chosen a 'greenfield approach' in designing the architecture. The proposed architecture does have many aspects in common with existing systems, however. Where appropriate, we have discussed relations to existing systems in the sequel of the paper. We have not taken the reference architecture of the Workflow Management Coalition (see Fig. 1) as a starting point, because we consider this architecture incomplete for this purpose (platform interfaces and data management are not explicitly included). The reference architecture described in this paper can, however, easily be related to that of the Workflow Management Coalition (this will be elaborated in Section 4.1).

3.2. Design principles

The workflow management system reference architecture is developed following a number of main principles for its design. These principles determine the 'design philosophy' of the architecture and guard the quality of the design. The design principles have been chosen to result in a reference architecture that allows for flexible comparison to and analysis of concrete workflow systems. Flexibility implies the possibility to view the architecture on several levels of detail on the one hand and with respect to several system aspects on the other hand. In analyzing workflow systems, the position of the workflow system in the overall system architecture is of paramount importance. Below, we discuss the four main principles, the first three of which contribute to the flexibility aspect, and the fourth of which aims at a clear positioning of the reference architecture in its technical context.

Firstly, a strict *top-down design strategy* is used that results in a clear multi-level, modular architecture design. The following design levels are to be distinguished: the WFMS in its system context, the global WFMS, the WFMS subsystems, and the WFMS subsystem modules. The modules can be further decomposed into module components, but this level is not within the scope of this paper. On each level, a modular architecture is described consisting of modules as well defined clusters of functionality with well-defined interfaces to other modules. Modules coincide with conceptual clusters of functionality. The modular architecture enables an incremental WFMS development strategy, based on requirements of new applications.

Secondly, a clear separation is made between *enactment and design perspectives* in the WFMS software architecture. Note that both perspectives are not completely independent, though. Both perspectives of the architecture operate on the workflow definition data: the design perspective creates these data, the enactment perspective uses these data. Further, they both make use of the same interface modules to access underlying software platforms. This separation of concerns allows for independent solutions that do not require any kinds of trade-offs between both aspects.

Thirdly, an explicit separation is made *between kernel functionality and additional functionality* for the WFMS. The kernel WFMS contains basic functionality necessary for all application types. Additional functionality is added to the kernel system by adding extension modules that can be connected to a software bus. Extension modules become integral parts of the WFMS, as opposed to the client/server approach to functional extensibility (see e.g. [8]). System extensibility is important for incremental system design and implementation (developer's perspective) and incremental system installation (end user perspective).

Finally, emphasis is put on *explicit interfaces* between the WFMS and the software platforms it uses. The use of dedicated interface modules for interfaces to software components used by the WFMS enables separation of concerns in architecture design and easy platform independence. Explicit interface modules have already been used in some WFM products, e.g. the Basic Access Layer of the FORO WFMS [10, 13]. In the architecture presented in this paper, interface modules are used consistently for all software interfaces, i.e. interfaces to operating system, database management system, communication system, user interface system, and application systems.

3.3. Architectural aspects

Apart from the general design principles outlined above, we discuss major architectural aspects that have also heavily influenced the design of the reference architecture.

Distribution is a major architectural aspect for a workflow management system for two reasons. Firstly, workflow management is by nature a distributed application in which many users at different places in an organization (or even several organizations) cooperate. Secondly, distribution opens the way to scalability. *Scalability* is of great importance to allow workflow applications to 'grow' in time because of increased number of users, number of cases, or size and complexity of cases.

Heterogeneity is a second major aspect for workflow management systems. Workflow systems typically operate in environments where servers and workstations of different kinds exist. Coupling these systems in enterprise-wide workflow systems requires an approach that can handle heterogeneous environments. When coupling multiple workflow management systems, heterogeneity can lead to improved *interoperability*.

4. Global WFM system architecture

This section describes the highest level of an abstract reference architecture for a WFM system. This architecture is further elaborated in the next sections to obtain an architecture specification with such detail, that it can be used for a high-level functional specification of a WFMS. As such, the software architecture specification is significantly more detailed than the one defined by the Workflow Management Coalition [43], shown in Fig. 1.

First, the global WFM system architecture is presented, discussing the top-level modules of the system. Then, the data sets manipulated by the WFM system are described and important issues with respect to data management are discussed. Finally, attention is paid to various interfaces in the architecture.

4.1. Global architecture

The global architecture of the WFM system in its environment is shown in Fig. 3. The WFM system itself consists of three main modules (having a dark gray color in the figure to indicate that they are decomposed in the sequel of the paper) providing workflow design, central enactment and end user functionality:

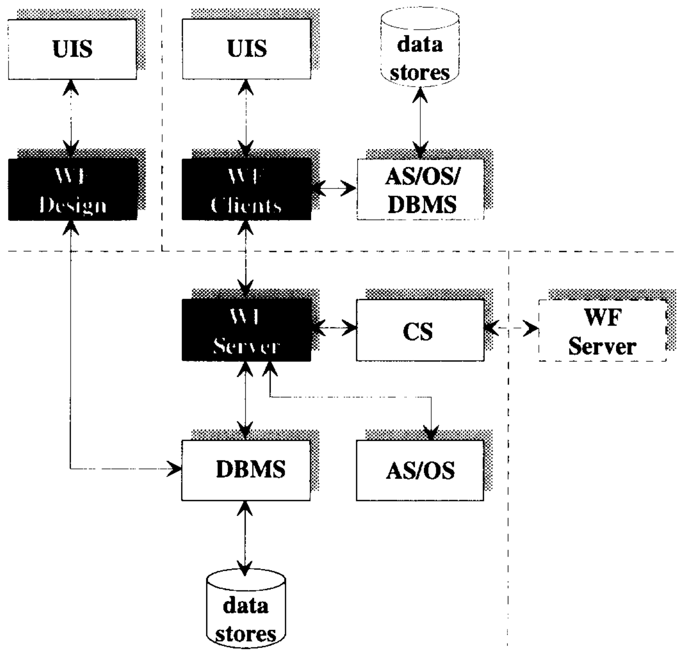

Fig. 3. Global WFM system architecture.

WF Design Module. The workflow design module provides services for the design of workflow applications. The design data is stored in the central data store and used by the workflow server modules. The design module is further elaborated in Section 5.

WF Server Module. The workflow server module provides all central services for workflow enactment. It communicates with workflow clients and possibly other workflow servers. The server operates on the data in the central data store. It is further elaborated in Section 6.

WF Clients Module. The workflow clients provide decentral end user services for workflow enactment and management. They communicate with the workflow enactment server. Enactment clients can also temporarily operate in a stand-alone fashion, using local data stores. The local data stores are synchronized with the central store using a check-in/check-out mechanism (see further Section 4.4). The client modules are discussed in detail in Section 7.

The WF server module uses the services of the communication system (CS) for information exchange with other workflow server modules in the case of distributed, cooperative applications. Application systems (AS) are used for application-specific functionalities, e.g. financial report generation in a financial workflow application. The operating system (OS) is used for low-level system operations, e.g. file system operations.

A complete workflow environment always contains the three main modules described above. There may be situations, however, where the workflow design functionality and the workflow enactment functionality are strongly separated into two independent architectures. This may be the case, for example, in a scenario where an IT-organization performs the complete workflow design in its own system context for an end-user organization. Clearly, both sub-architectures have to operate on compatible data stores in this situation to enable transfer of workflow application from developer to user context.

Note that the three modules often run in a heterogeneous environment, in which the various modules use different operating platforms. The boundaries between these platforms are indicated by the dotted lines in Fig. 3. The heterogeneous environment may require partial replication of data stores; this aspect is not dealt with here for reasons of clarity.

The relation between the global architecture in Fig. 3 and the WFM Coalition standard architecture in Fig. 1 can be easily explained. The Workflow Engine in the WFMC standard architecture coincides with the WF Server module in Fig. 3. The interfaces in the WFMS architecture are related to those in Fig. 3 as follows: IF1 is the interface between WF Design module and DBMS, IF2 and IF5 are the interface between WF Server and WF Clients, IF3 is the interface between WF Server and AS/OS, IF4 is the interface between WF Server and CS. The fact that IF2 and IF5 coincide in our architecture follows from the fact that administration and management tools are considered special-purpose clients.

4.2. Interfaces in the global architecture

In the architecture shown in Fig. 3, a number of important interfaces can be distinguished. Below, we discuss the interface to the DBMS, the interface between workflow server and client modules, and the interface between multiple workflow servers via the communication system. Further relevant details on interfaces are included in the sequel of this paper.

Both the WF design module and the WF server module use a DBMS for persistent data storage. The design module uses the DBMS at workflow definition time to store workflow definitions, the server module at workflow enactment time to read workflow definitions and store workflow data. If both modules are allowed to access workflow specifications concurrently, the situation can occur where a workflow definition is modified that is also in execution. To handle inconsistencies in this workflow evolution situation, a versioning mechanism is required in the server module.

The design and server modules use the DBMS in very different ways: the design module performs infrequent high-volume accesses to the database, the server module frequent relatively low-volume accesses. As the server module is performance-critical, the DBMS interface should be tuned towards its usage pattern. As the vast majority of commercial DBMSs uses a relational data model, the interface is mostly of the relational type, i.e. based on the SQL standard. This means that complex data structures used in design and server modules must be mapped to the relational model. This is a common approach used for example in the InConcert product [30] and the WIDE project [10,13]. Details on the DBMS interfaces included in the modules are given in Section 8.2.

The interface between the workflow server and workflow client modules is used to pass workflow events and related information between the central server and a number of decentral clients. Communication is usually of a small grain size, but may be of a larger grain size in the case of a 'thick' client (see Section 7.1). As workflow server and client usually belong to the same software suite, the interface is usually of a dedicated nature, based on a client/server protocol. Communication standards like CORBA [36] or HTTP can be used, however, to obtain some standardization in the infrastructure of this interface. The HTTP standard can be used for example to allow the use of clients based on Java [4] that operate in the environment of WWW-browsers (an intranet approach to workflow management).

The interface between multiple workflow server modules (via the communication system) is used to support workflows that span multiple workflow servers. In principle, one can distinguish two 'ambition levels': process invocation and process transfer. In the former case, the interface is used to simply invoke process definitions at a remote site, possibly passing parameters. In the latter case, process definitions are passed through the interface to be executed by a remote server. This situation can be based for example on the WfMC process interoperability standard [44].

4.3. Workflow data sets

In the global WFMS architecture, a set of data stores is included for storage of workflow data and meta-data (see Fig. 3). In these data stores, a number of data sets is distinguished for workflow design and enactment. The schemas (and partly the contents) of these data sets depend heavily on the conceptual model used for workflow applications, i.e. on the way workflow entities and relations between them are modeled (see e.g. [10] for an elaborated conceptual model).

The context of workflow applications is described in two data sets. The *organization data set* contains information about the organizational context of workflow applications, like organization structure, function descriptions, personnel descriptions, resource descriptions, etc. The *product definition data* set contains detailed descriptions of the products of the organization, the production of which is supported by the workflow management application. The descriptions can include information on necessary resources for the production of the products.

The design of workflow applications is described in the *workflow definition data set.* It contains the information on the design and structure of workflow applications, like workflow specifications and task specifications. Often, this information is extracted or compiled from workflow specifications in some workflow specification language (see e.g. [19,15]). The data in the workflow definition data set can be considered the meta-data of the workflow system.

Workflow enactment data is stored in three data sets. The *workflow process data set* contains information on the current status and history of workflow applications, e.g. the status of tasks being performed or having been performed. The data set may be stored in a distributed fashion on multiple databases on server (e.g. to store central process data) and client machines (e.g. to store local inboxes and outboxes). The *management information data set* contains aggregated workflow process data for management information and evaluation purposes. The data is periodically updated by extracting relevant data from the process data set. The data set is used as a data warehouse on which historical queries can be formulated. The *application data set* contains the workflow case data (objects) actually processed by the workflow application or descriptions thereof. Workflow cases can be seen as intermediate products in the production of the products described in the product definition data. Note that the application data may also be accessed directly by application systems.

4.4. Workflow data management

Workflow data sets are centrally managed by the workflow server (through one or more database management systems, see Section 8.2). It may be necessary, however, to temporarily move parts of the data to local databases on client platforms. An operational reason can be to enhance performance by local caching of relevant data on client stations. A functional reason is the necessity of off-line processing of certain workflow tasks. This situation can occur in geographically distributed workflow management applications where data connections are not available in a continuous fashion. The necessity also arises in the context of mobile workflow clients, a situation that is interesting in application areas where workflow management extends beyond the traditional office environment (see e.g. [33]).

To enable data caching at client sites, specified parts of the data sets can be checked out of the central data store to allow decentral autonomous processing of parts of workflows using local databases. After this processing, the data is checked in again into the central database, during which attention is paid to possible consistency and versioning problems with the data. Mechanisms for check- out and check-in of data have already been described in other application areas of databases having comparable transaction management characteristics (see e.g. [27,26,16,17]).

5. Workflow design module architecture

This section presents the system architecture for the design perspective of the WFM system, i.e. the WF design module from Fig. 3 and its environment. First, the global architecture of the design module is discussed, in which the design engine is the central component. Next, the architecture of the design engine is described in more detail. Finally, extension modules are discussed that can be used to enhance the basic functionality of the design engine.

5.1. Global design module architecture

The basic WFM system architecture for workflow design is depicted in Fig. 4. We can distinguish the following modules:

WF Design Engine. The workflow design engine provides basic workflow design tools. The various components of the engine are discussed in detail in Section 5.2.

UIS/AS/DBMS Interface. The interface modules provide interfaces to UIS/AS/DBMS to hide

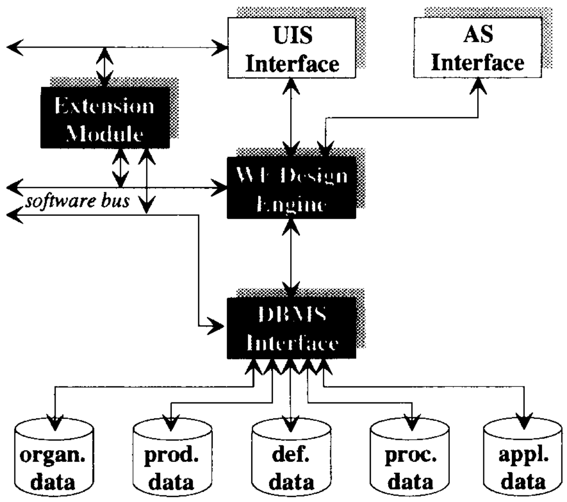

Fig. 4. Workflow design module architecture.

platform details. Note that the DBMS interface accesses data sets through the DBMS (not shown in the figure for clarity).

Extension Modules. One or more extension modules can be used to enhance the functionality of the design engine. Modules are connected to the engine through a software bus (see Section 8.1). A set of extension modules is discussed in detail in Section 5.3.

Design engine and extension modules are described below in more detail.

5.2. Design engine architecture

Workflow design is a complex process with a number of steps, each of which requires attention to different aspects of workflow models and specifications. A well defined design process (also refered to as way-of-working [34]) is therefore an important aspect of workflow design. The architecture of the workflow design engine is consequently derived from the various steps that can be distinguished in the workflow application design process.

In this process, the organization in which the workflow is to be embedded and the products to be supported by the workflow are defined first. The organization and product specifications set the context for the actual workflow specification.

In the workflow specification, a global design of the workflow application is constructed first by senior application designers in cooperation with functional managers. The emphasis in this phase is usually on the global workflow process structure.

The global design is subsequently further elaborated by expert application designers to result in a complete specification of the application. In this phase, process designs are further refined, and workflow details like workflow tasks contents, task allocation rules, scheduling information, and exception handler routines are specified

This specification can next be enacted (executed). During enactment of the application, tuning is performed on detail aspects by operational workflow managers. The order of the design steps is illustrated in Fig. 5.

In accordance with the five design steps, the workflow design engine consists of five main interactive design modules that all operate on the same meta-data and one interface module, as depicted in Fig. 6. These modules are discussed below:

Organization specification module. The organization specification module is used by functional (strategic) managers to specify the organization structure, roles, and resources with respect to workflow management. The module modifies the contents of the organization data set.

Product specification module. The product specification module is used to specify products and sub- products to be used in workflows. An example of a product is an electronic form to be routed in a workflow. The module modifies the product data set.

Global design module. Given organization and product characteristics, the global design module is used to set up the global design of workflow applications. The user interface of the module should be such that it is understandable for non-IT-experts, e.g. a graphical interface to define the process structure. The module modifies the workflow definition data set. The organization and product data sets are used as input for the design process. Process and application data sets are used as input if information on the current application status is necessary (e.g. for simulations).

Detail design module. The detail design module is used to further elaborate global workflow application designs. It offers mechanisms to refine the global process design constructed with the

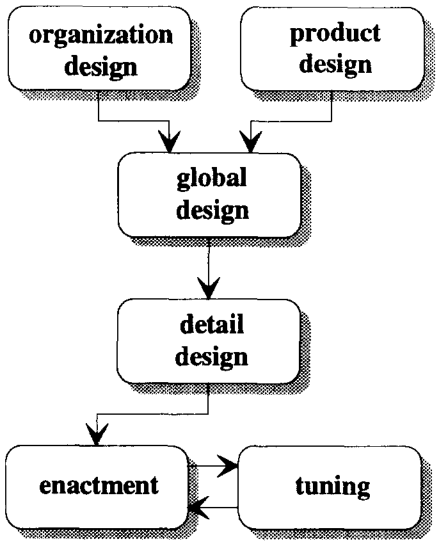

Fig. 5. Workflow design phases.

global design module. It also offers possibilities to specify workflow details, e.g. through a form-based interface. This module delivers completely specified and executable workflow applications. The module uses the data sets in the same way as the global design module. The detail design module may produce a specification that is not directly fit for enactment by the workflow engine. In this case, the enactment code has to be produced by a compiler or generator. To allow flexibility in the configuration, we see this as a design extension module (discussed in Section 5.3).

Tuning module. The tuning module is used to perform small modifications to existing workflow applications, like local changes in task allocation. As such, the module is not used to change workflow

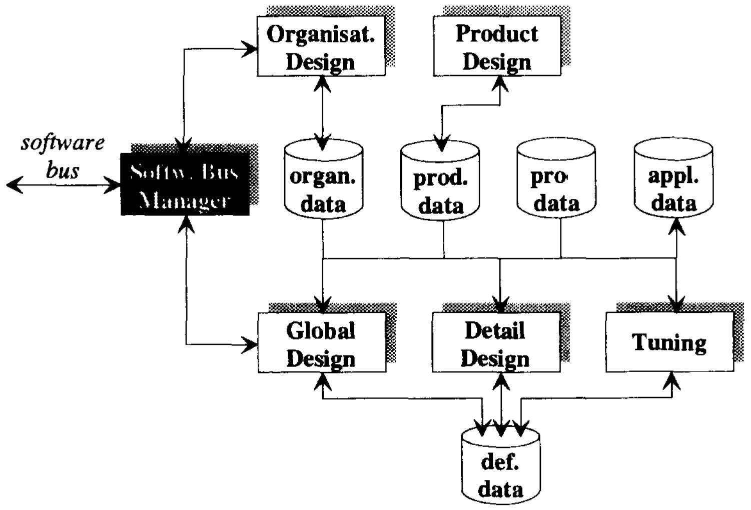

Fig. 6. Workflow design engine architecture.

process structures, but rather to adapt process parameters where this turns out to be necessary during workflow enactment. The module provides limited functionality with respect to detail design. The module modifies the workflow definition data set to reflect modified parameters. Other data sets are used in a limited fashion as described for the two above modules. Clearly, the tuning module should not require the use a code generator as described above (small modifications should not require extensive regeneration of enactment code, but rather modify some variables used by this code).

Software bus manager. The software bus manager is an internal component, providing the interface to design extension modules on the software bus. This module is connected to all design modules (for reasons of clarity, not all connections are shown in the figure).
Note that it is possible to combine various design modules, e.g. having one module providing the functionality for global design, detail design and tuning. We believe however that the various design tasks are that different in nature that different modules are a more adequate solution.

Further details on the functionality of the design engine modules can be found in [32].

5.3. Design extension modules

As discussed before, extension modules can be added to the basic workflow design architecture to support the workflow design process in specific ways. Below, a few extension modules are discussed; a more complete list can be found in [23].

Extension modules can be used for the static verification of defined workflows. A *syntax checker* module is used to check the syntax of the workflow application components, i.e. the organization, product and process models. The module performs syntactical analysis on aspects like completeness (e.g. of attributes) and structure (e.g. circular relationships). Further, the module checks cross-references between the various models. A *semantics* checker module can be used to analyze the semantics of defined workflow models. Example aspects are: dead ends in workflows, cycles in workflows, specified resources that are not actually used in the models, and unreachable paths in workflows.

Depending on the format of the workflow specification produced by the detail design module and the format required by the workflow enactment engine, a compilation or generation of enactment code may be necessary. This task is performed by a *code generator* extension module. Having this as module as an extension module allows for flexibility with respect to server platforms. In the WIDE system [10], a compiler is used to translate the WWPDL language into contents of relational tables used by the workflow engine. In the Exotica project, a generator is used to add advanced rollback functionality to workflow processes [1].

Another important use of extension modules is the dynamic analysis of defined workflows. A *workflow simulator* module is used to dynamically simulate defined workflow applications to provide insight in the dynamic behavior of these applications. The module can access application data in the database to obtain simulations that are closely related to the actual situation. The simulator can be based on a more general dynamic simulation environment like ExSpect [5] or Arena [40].

Extension modules can also aid in the management of workflow definitions. A *workflow library manager* module is used by workflow application designers to manage a library of reusable standard workflow process blocks. The reuse of workflow 'building blocks' improves the efficiency and quality of the workflow design process.

6. Workflow enactment server architecture

This section discusses the architecture of the workflow enactment server, i.e. the central module in workflow execution (see Fig. 3). The workflow enactment clients that communicate with the server are discussed in Section 7.

6.1. Global enactment server module architecture

Below, the global enactment server architecture as shown in Fig. 7 is described. We can distinguish the following modules in this architecture:

WF Server Engine. The workflow server engine provides basic enactment services by interpreting workflow specifications in the workflow definition data store. It records activities into the process data store. It can access application data where necessary, e.g. for content-based routing. The engine can be operated in dummy (for testing purposes) and production modes. The engine is discussed in detail in Section 6.2.

WF Client Interface. This interface module provides the interface between the workflow server and workflow management clients (see Section 7) in the client/server architecture.

CS Interface. The CS interface module is used to communicate with other workflow servers. It shields platform-specific details of the communication infrastructure. For its communication, it preferably uses a language based on a standard like the WfMC interoperability proposal [44].

AS/OS/DBMS Interfaces. The platform interface modules provide standard interfaces to AS/OS/DBMS platforms by hiding platform-specific details.

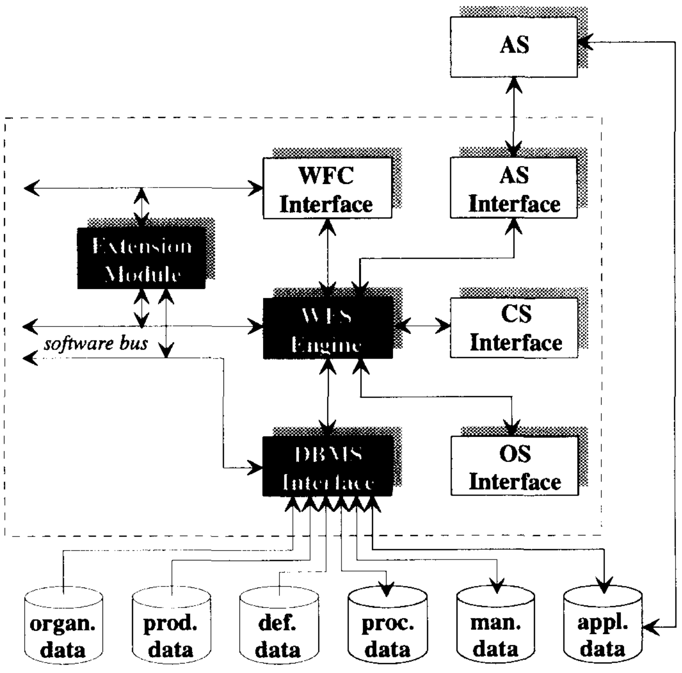

Fig. 7. Workflow enactment server architecture.

Extension module. The extension modules provide additional functionality to the basic workflow engine. Extension modules can be added in arbitrary numbers using the software bus. The actual selection of extension modules depends on the application domain at hand. A set of enactment server extension modules is discussed in Section 6.3.
In the above architecture, the workflow enactment engine is the central component. This engine is described in more detail below. Next, extension modules are described that can be used to enhance the functionality of the basic enactment engine.

6.2. Enactment server engine architecture

The main function of the enactment server engine is processing workflow events, i.e. analyzing events and initiating the reactions to these events as described in the workflow specification. As such, it implements a chosen basic workflow processing model. This model can be extended by the functionality of extension modules, as discussed below.

The workflow enactment server engine architecture is shown in Fig. 8. It is based on a phased processing of workflow events, allowing a high degree of flexibility in this processing. The subsequent phases of event processing are event reception, event analysis, action synthesis, and action execution. Each phase is associated with a specific engine module. Adding two support modules, the following modules can be distinguished in the architecture:

Clock module. As time events are important in most office applications [35], an explicit clock

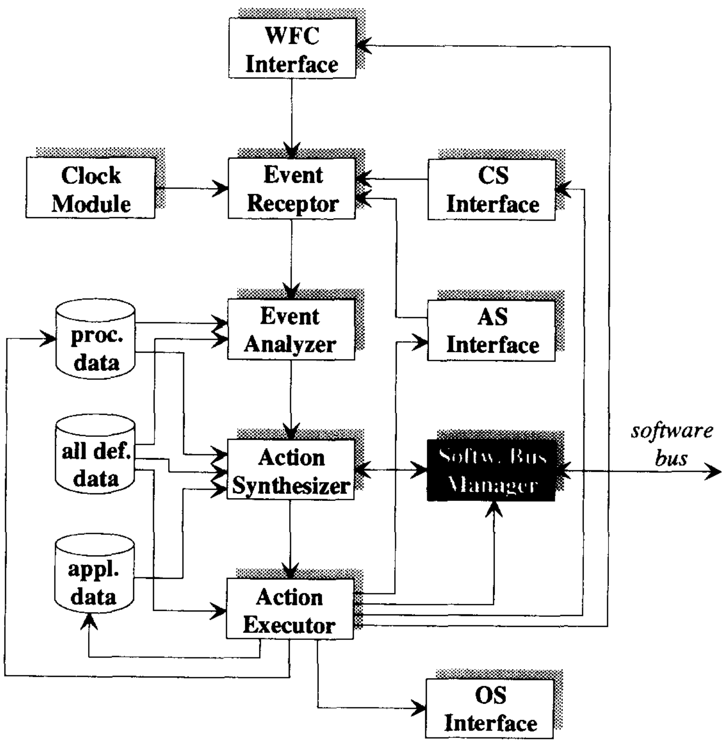

Fig. 8. Workflow enactment server engine architecture.

module is included. The clock module generates clock events for the initiation of time-triggered workflow actions. The clock event granularity and/or clock event priorities can be adapted based on information about the current system load.

Event receptor. The event receptor module receives all events from other modules and normalizes them. Event classes are clock events, application system events, communication system events, and workflow client events. Events are normalized based on event classes and event sources. Specific event classes can be enabled and disabled, providing a filtering mechanism for events. Normalized events are prioritized and queued for further processing by the event analyzer module.

Event analyzer. This module analyzes incoming events on the one hand and cases in the process data store on the other hand to select all case identifiers to be processed on a specific event occurrence. Cases are selected from process data using event indices attached to the cases. Cases can be prioritized based on flexible combinations of case and event priorities. The module queues cases for further processing by the action synthesizer module.

Action synthesizer. The action synthesizer module constructs actions to be performed on selected cases based on process definition data, process data, and application data. It can be seen as the module that interprets state-transition diagrams representing the life cycle of workflow cases (see for example [10]). It analyzes cases and their processing context (e.g. resources and related cases) and evaluates routing conditions based on process, definition, and application data. Actions to be performed are prioritized and queued for further processing by the action executor module. The action synthesizer uses two-way extension modules if available as enhanced replacements of internal functionality.

Action executor. The action executor module is responsible for the execution of actions constructed by the reaction synthesizer module. It accesses and modifies process and application data as necessary. Where necessary, it invokes end user applications. It updates process data, e.g. the inboxes of workflow clients. It communicates with workflow clients through the workflow client interface. Where communication with other workflow enactment services is necessary, it invokes CS actions. If appropriate, the action executor communicates with one-way extension modules.

Software bus manager. The software bus manager provides a uniform interface to extension modules on the software bus. It communicates with server modules and extension modules on the software bus. Requests for and answers from extension modules are queued for processing. The module administrates availability of two-way extension modules and activates one-way extension modules available. It is discussed in more detail in Section 8.1.

Note that all definition data sets (organization, product, workflow) are shown as one data set (labeled 'all def. data') in Fig. 8 for reasons of clarity.

The above architecture presents the functionality from a server process oriented view, i.e., the workflow management functionality is located in the server process which operates on the workflow data. From an object-oriented point of view, part of the functionality may be located in workflow case objects that conceptually contain both the workflow data pertaining to a workflow case plus some of the functionality working on that data.

6.3. Enactment server extension modules

The basic workflow enactment server can be extended with extension modules that add additional functionality to the basic server. The addition of extension modules is application-dependent: the higher the functional requirements of a workflow application environment, the more extension modules will be required.

We distinguish between two types of extension modules based on the communication pattern with the workflow server engine. *Two-way extension modules* are fed with information from the basic architecture and produce information for the workflow server engine. As such, they are extensions or replacements of internal functionality of the workflow server engine. *One-way extension modules* are fed with information from the basic architecture but do not return information to the engine. Their task is to provide information services to external entities like end users or workflow system administrators. Extension modules can further be classified according to their interaction with their users: no interaction at all, interaction through standard client mechanisms, or interaction through a dedicated client.

Important examples of two-way extension modules are modules to intelligently handle resource and task allocation, and exceptions occurring in workflow enactment. An *intelligent resource manager* module manages resource allocation to workflow tasks to obtain optimal usage of scarce resources or minimal waiting times. Resources can be equipment like a fax machine or a printer, or office space like a meeting room. The module can use logistic optimization algorithms to compute resource allocations. A *policy resolution manager* module is used to resolve complex task assignment policies according to predefined business rules (see e.g. [8]). It tries to find the best possible actor (office worker) to accomplish a given task, taking into account complex information like functions, tasks, availability, replacement rules, and authorizations of actors. An *intelligent exception handler* module handles exceptional situations in workflow enactment, as defined in business rules. The module may be based on a rule (expert) system that implements a set of business rules. It possibly communicates with an authorized user in case of undefined or critical situations to find the best response to an exceptional situation.

One-way extension modules often provide information to workflow actors or managers. The workflow monitor module provides management and control information on workflow execution, e.g. average waiting times, average processing times, etc. It uses a dedicated client for presentation to the user. This client is capable of displaying the information such that it easily supports operational decision making. The *management information provider* module provides aggregated management information for workflow application evaluation and tactic/strategic decision making.

A more complete list of enactment extension modules can be found in [23].

7. Workflow client architecture

This section discusses the workflow enactment client architecture, i.e. the architecture of the decentral system modules for workflow enactment. Several client module types can be distinguished:

Workflow enactment client. The workflow enactment client provides the interactive functionality by which actors (office workers) can perform their tasks in the workflow application. This client often uses an inbox/outbox paradigm for the presentation of jobs to be performed by actors.

Workflow administrator client. The workflow administrator client offers functionality to manually control workflows where necessary, e.g. to remove or reroute specific cases. This client is used by a workflow supervisor or manager.

Extension module client. Extension module clients are dedicated client modules used by extension

modules as discussed in Section 6.3. Examples are the workflow visualizer client, a graphical client module used by the workflow visualizer extension module, and the workflow monitor client, a client module used by the workflow monitor extension module.
In the context of this paper, only the workflow enactment client is interesting from an architectural point of view, as the other client types have very simple architectures. Below, the global architecture of the enactment client is described first. Next, attention is paid to the detailed architecture of the local enactment engine, which is the 'heart' of the enactment client.

7.1. Enactment client architecture

The workflow enactment client architecture is shown in Fig. 9. The architecture provides the functionality required by end users to perform their tasks in workflow applications.

The following components are distinguished in the enactment client architecture:

WF Engine. The client workflow engine provides local workflow enactment services, enabling delegation of specific enactment tasks by the central enactment server. These enactment tasks always have a scope local to the engine (usually one actor). The client engine can operate in various modes with respect to the delegation by and communication with the central workflow server. These modes are discussed in detail below.

WFS Interface. The workflow server interface provides the interface between the local workflow client engine and the central workflow enactment server. It is also responsible for the check-in and check-out of data as discussed before in this paper. Its function depends on the mode in which the local workflow engine operates.

UIS Interface. The UIS interface provides specific workflow client functionality that has an interactive nature. The most prominent functionality is the handling of workflow inboxes and outboxes through which the end users receive and submit workflow tasks. Electronic form handling may also be part of the UIS interface functionality. It hides platform specific user interface details from the local workflow engine and the workflow server interface.

AS/OS Interface. These interface modules provide interfaces to AS and OS to hide platform specific details. The AS interface is used for activating and controlling interactive applications used by the end user in the workflow application, like text processors, spreadsheet programs, etc. The OS interface is used to provide access to low level platform functionality, e.g. file handling.

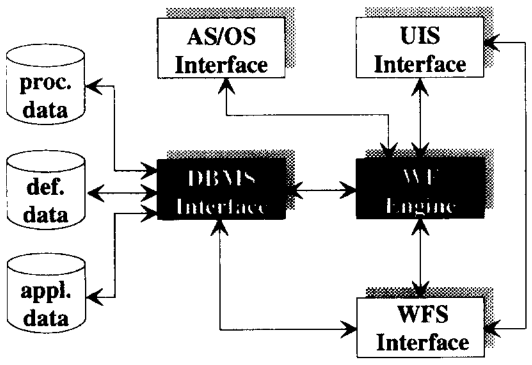

Fig. 9. Workflow enactment client architecture.

DBMS Interface. The DBMS interface module provides transparent access to both central workflow server database (using the workflow server interface) and local workflow client database, depending on the mode in which the local engine operates (see below).

Note that a communication system interface module is not included in the client architecture, because all external communication (beyond the scope of the client) is performed through the central workflow server.

The workflow client can operate in three main modes, determining the way the client interacts with the workflow server. In *direct synchronous mode*, the workflow server interface communicates directly with the user interface system interface. The activities of the client are directly controlled by the central workflow server. The local client workflow engine and database system are not used in this mode. In *cached synchronous mode*, the workflow client operates with a copy of part of the central workflow data. Local and central data are always completely synchronized at relevant points, i.e. the local data store is a cache for the central data store. The activities of the client are mainly controlled by the local workflow engine. In *stand-alone asynchronous mode*, the workflow client operates with a local copy of the central workflow data that is not necessarily synchronized at all moments. Synchronization is performed at specific points in time using check-out/check-in protocols between local and central databases (see Section 4.4). The activities of the client are completely controlled by the local workflow engine. A client that operates in direct synchronous mode can be called a 'thin' client, whereas the other two modes require a 'thick' client.

The choice for a specific mode depends on the characteristics of the environment in which the workflow system is used. The direct synchronous mode is the simplest mode of operation, used in tightly coupled environments in which the central workflow server has adequate performance to handle all tasks. The cached synchronous mode is useful to reduce communication between server and client modules in tightly-coupled environments with high workloads for the central server. The stand-alone asynchronous mode is useful when an on-line connection between central server and decentral clients is impossible or very costly, e.g. in the case of mobile clients.

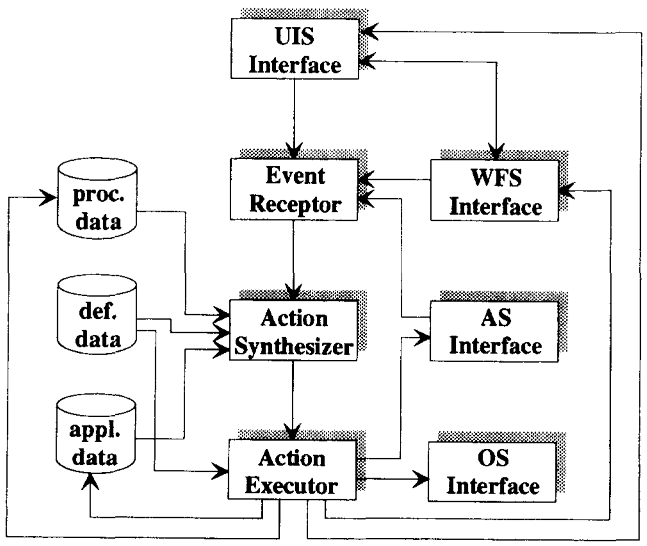

Fig. 10. Workflow enactment client engine architecture.

7.2. Enactment client engine architecture

The workflow enactment client engine architecture is shown in Fig. 10. The architecture can partly be seen as a simplified form of the enactment server engine as depicted in Fig. 8, with the following main differences.

The client engine is not equipped with an event analyzer module, because the limited scope of the local workflow process does not require advanced analysis to determine the cases to be processed (often, the client is processing one single case). Because of its limited functionality, the client engine is not equipped with a software bus manager module. A clock module is not part of the client engine architecture because advanced timing functionality is not required for the client; clearly, the module can easily be added if necessary.

8. Interfaces

In this section, attention is paid to the interfaces between the kernel WFMS and its environment, the software environment described in Section 2 and the extension modules discussed in Sections 5.3 and 6.3. As discussed before in this paper, the interfaces have each been mapped to interface modules to obtain platform independence and extensibility. As a detailed description of all interface modules is beyond the scope of this paper, we focus on two important interface modules: the software bus manager interface and the DBMS interface.

Work on application and communication system interfaces has already been presented elsewhere. A proposal for an application system interface is presented for example in [39]. Here, software wrapping techniques are used to standardize the interface to application systems. Part of the communication system interface is covered by the interoperability standard currently worked on by the WfMC [44]. Further, standards from the field of EDI can be used here.

8.1. Software bus interface

As described in the previous chapters, the software bus manager is used to control the software bus to which extension modules can be connected to extend the functionality of the basic workflow management system.

The internal architecture of the software bus manager is shown in Fig. 11. The *protocol manager*

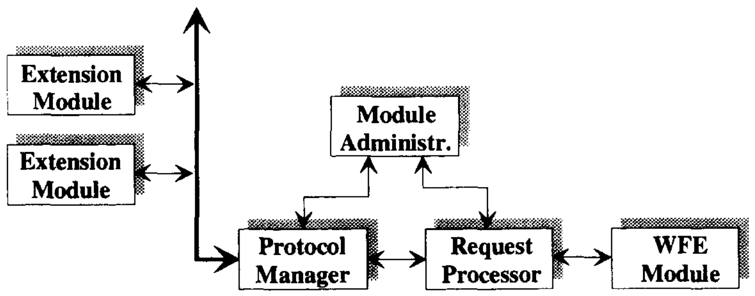

Fig. 11. Software bus manager architecture.

module is responsible for the communication protocols on the software bus, i.e. for the communication of requests and results, the handling of exceptions, etc. The protocol manager directly communicates with the extension modules over the software bus. The specification of software bus communication protocols are not within the scope of this paper. The *request processor* module accepts request from the workflow engine (WFE) modules and queues them for processing by the extension modules. The module is also responsible for delivering results from two-way extension modules to the appropriate basic system module. The *extension module administration* module registers the extension modules present on the software bus.

To obtain a dynamically configurable system, the software bus interface can be based on a dynamic interface protocol, e.g. CORBA's Dynamic Invocation Interface (DII) which allows dynamic registration and use of interfaces [36].

8.2. DBMS interface

The database management system (DBMS) is the repository for data and meta-data of the workflow management architecture, as described in Section 4.3. Depending on the type of DBMS, it offers specific functionality with respect to supported data structures, definition and query languages, transaction support, etc. Further, multiple DBMSs may be used in a workflow management application environment. Finally, the workflow system may use non-DBMS systems for data storage. An example of this situation has been investigated in the Exotica Project, in which the Flowmark WFMS and the Lotus Notes replicated file system have been coupled [2]. The DBMS interface module is designed to hide platform specific details of data management systems from the workflow management system.

The architecture of the DBMS interface is depicted in Fig. 12. A separation is made between data access functionality and transaction support functionality (see [13] for a comparable approach for the FORO DBMS).

The *high-level transaction manager* module is responsible for the management of advanced

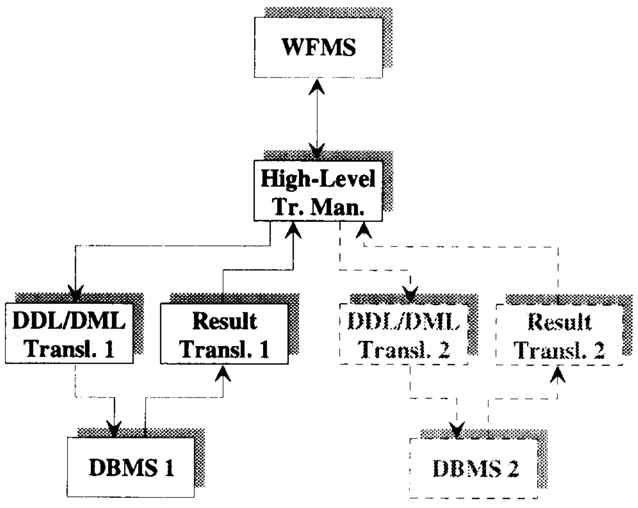

Fig. 12. DBMS interface architecture.

transaction classes not supported by the underlying database system(s) (see e.g. [16]). The module provides additional transactional primitives with respect to the underlying database platform, comparable to the transaction adapters in [6]. It provides high-level transaction management on top of standard transaction management offered by database platforms. Important transaction classes for workflow management are distributed transactions and long-lived transactions. For the class of distributed transactions, the high-level transaction manager must provide a two-phase commit protocol over local transaction managers [12]. For the class of long transactions, it may provide a saga mechanism offering undo functionality through compensating transactions [20]. A more complex approach has been developed in the WIDE project, in which the high-level transaction manager consists of two levels itself offering two levels of transaction functionality on top of the transaction mechanism of a commercial DBMS [13,24].

The *DDL/DML translator* module is responsible for the translation of data definition language and database manipulation language constructs as used by the workflow management system to the specific languages of the underlying database system(s). The *result translator* module translates query results from the database system(s) into the standard internal format used by the workflow system. The formats offered by both translator modules should preferably be based on commonly accepted standards, like CORBA [36] or ODMG [11]. In the WIDE project, a Basic Interface Layer is used to perform the functions of the translator modules [10,13]. This BAL module provides an object-oriented C++ interface to its clients and uses a relational interface to the underlying commercial DBMS.

A number of design questions have to be answered for the interface between the WFMS and the DBMS. These questions are related to the data related interface between workflow system and database system and to the process-related interface between both systems. The primary question related to the data interface concerns the data model offered by the DBMS interface module to the various WFMS modules and the data model(s) that the DBMS supports. If the internal format is object-oriented and the DBMS is relational, for example, the interface module should offer a conversion from object-oriented language constructs to relational language constructs (DDL/DML translator) and a conversion from relational query results to object-oriented constructs (result translator). The primary question concerning the process interface is related to the DBMS operation interface granularity and the DBMS activity type. The DBMS operation interface granularity determines the kind of operations that are sent from workflow system to database system: object-operation, set-operation, basic transaction (see e.g. [30] for a discussion of this issue in the context of the InConcert WFMS), or extended transaction (see [16] for an overview of extended transaction models, [24] for the application of extended transactions in the context of the WIDE WFMS). A second important question concerning the process interface is whether the database system is a passive server (traditional DBMS) or an active server (ADBMS, see e.g. [22]). If the database system is a passive server, all initiative for actions lies within the workflow management system. If the database system is an active server, it can trigger actions itself and cooperate with the workflow system or even control the actions of the workflow system.

9. Conclusions

This paper discusses a reference architecture for an 'ideal' workflow management system. The system is called 'ideal' because it has been designed using clear design principles, the objective has been to describe complete functionality, and no limitations have been present related to existing systems. In the development of the architecture, specialists from various backgrounds (research, consultancy and end-user) have contributed, thus improving the completeness of the design.

The workflow management system architecture presented in this paper goes into deeper detail and is more complete than other existing standard architectures. In the first place, the described architecture can be used as a reference architecture to analyze and compare existing workflow management systems, e.g. with respect to functional completeness or interoperability. In the second place, the architecture can be used as a solid basis for the development of new workflow management systems or modules thereof. Further, the reference architecture can be of help in the configuration of heterogeneous modular systems, i.e. architectures that have been configured from modules with different origins.

One obvious direction of possible further work is the realization of a prototype of the architecture described in this paper. Given the strict modularity of the architecture, the realization can be limited to a selected set of modules in the architecture, depending on the required functionality. A second direction of possible further work is the detailed specification of the internal and external interfaces described in the architecture. Descriptions of the interfaces are useful in the first place to obtain interoperability between heterogeneous workflow management systems. They are also essential to the development of heterogeneous modular architectures as mentioned above.

Acknowledgements

All participants in the Mercurius project are thanked for their comments and remarks with respect to the ideas presented in this paper. Special thanks go to Pieter Canton of Rabobank Nederland for his input from the 'practical field' of workflow management. Ria Coonjers is thanked for advice on textual matters.

References

[1] G. Alonso, D. Agrawal, A. El Abbadi, M. Kamath, R. Günthör, C. Mohan, Advanced transaction models in workflow contexts, *Procs. Int. Conf. on Data Engineering*; (1996).

[2] G. Alonso, B. Reinwald, C. Mohan, Distributed data management in workflow environments, *Procs. 7th Int. Workshop on Research Issues in Data Engineering*, Birmingham, UK (1997).

[3] G. Alonso, D. Agrawal, E. El Abbadi, C. Mohan, Functionality and limitations of current workflow management systems, *IEEE Expert* 12(5) (1997).

[4] K. Arnold, J. Gosling, *The Java Programming Language* (Addison-Wesley, Reading, MA, USA, 1996).

[5] *ExSpect: Executable Specification Tool Based on Formal Modeling Techniques* (Bakkenist Management Consultants, Diemen, The Netherlands, 1995).

[6] R. Barga, C. Pu, A practical and modular method to implement extended transaction models, *Procs. 21st Int. Conf. on Very Large Data Bases*, Zurich, Switzerland (1995).

[7] Bellcore, The Bellcore OSCA architecture, *Technical Reference TR-STS- 000915*, Issue 1 (Bellcore, October 1992).

[8] C. Bussler, Policy resolution in workflow management systems, *Digital Technical Journal* 6(4) (1994).

[9] G. Canals, N. Boudjlida, J. Derniame, C. Godart, J. Lonchamp, ALF: A framework for building process-centered software engineering environments; In A. Finkelstein, J. Kramer, B. Nuseibeh (eds.), *Software Process Modelling and Technology* (Research Studies Press, Taunton, Somerset, England, 1994).

[10] F. Casati, P. Grefen, B. Pernici, G. Pozzi, G. Sánchez, WIDE: Workflow model and architecture, *CTIT Technical Report 96-19* (University of Twente, 1996).
[11] R.G.G. Cattell (ed.), *The Object Database Standard: ODMG-93* (Morgan Kaufmann, San Mateo, CA, USA, 1993).
[12] S. Ceri, G. Pelagatti, *Distributed Databases: Principles and Systems* (McGraw-Hill, New York, USA, 1984).
[13] S. Ceri, P. Grefen, S. Sánchez, WIDE—A distributed architecture for workflow management, *Procs. Workshop on Research Issues in Data Engineering*, Birmingham, UK (1997).
[14] D. Cerutti, D. Pierson, *Distributed Computing Environments* (McGraw-Hill, New York, USA, 1993).
[15] D. Chan, J. Vonk, G. Sánchez, P. Grefen, P. Apers, A conceptual workflow specification language, *CTIT Technical Report 96-48* (University of Twente, 1996).
[16] A.K. Elmagarmid (ed.), *Database Transaction Models for Advanced Applications* (Morgan Kaufmann, San Mateo, California, USA, 1992).
[17] R. Elmasri, S.B. Navathe, *Fundamentals of Database Systems*, 2nd ed. (Benjamin/Cummings, Redwood City, CA, USA, 1994).
[18] A. Finkelstein, J. Kramer, B. Nuseibeh (eds.), *Software Process Modelling and Technology* (Research Studies Press, Taunton, Somerset, England, 1994).
[19] A. Forst, E. Kühn, O. Bukhres, General Purpose Work Flow Languages, *Distributed and Parallel Databases* 3(2) (Kluwer Academic Publ., 1995).
[20] H. Garcia-Molina, K. Salem, Sagas, *Procs. 1987 ACM SIGMOD Int. Conf. on Management of Data*, USA (1987).
[21] D. Georgakopoulos, M. Hornick, A. Sheth, An overview of workflow management: From process modeling to workflow automation infrastructure, *Distributed and Parallel Databases* 3(2) (Kluwer Academic Publ., 1995).
[22] P.W.P.J. Grefen, Concepts and techniques for an active database approach to office procedure support, *Memorandum INF93-25* (University of Twente, 1993).
[23] P.W.P.J. Grefen, R.N. Remmerts de Vries, A reference architecture for workflow management systems, *Memorandum INF95-36* (University of Twente, 1995).
[24] P. Grefen, J. Vonk, E. Boertjes, P. Apers, Two-layer transaction management for workflow management applications, *Procs. 8th Int. Conf. on Database and Expert System Applications* (Toulouse, France, 1997).
[25] *Information Warehouse Architecture I*, SC26-3244 (IBM Corporation, 1993).
[26] R.H. Katz, *Information Management for Engineering Design* (Springer-Verlag, Berlin, Germany, 1985).
[27] W. Kim, R. Lorie, D. McNabb, W. Plouffe, A transaction mechanism for engineering design databases, *Procs. 10th Int. Conf. on Very Large Data Bases*, Singapore (1984).
[28] N. Krishnakumar, A. Sheth, Managing heterogeneous multi-system tasks to Sspport enterprise-wide operations, *Distributed and Parallel Databases* 3(2) (Kluwer Academic Publ., 1995).
[29] F. Leymann, D. Roller, Business process management with flowmark, *Procs. 39th IEEE Computer Society Int. Conf.* (San Francisco, USA, 1994).
[30] D.R. McCarthy, S.K. Sarin, Workflows and Transactions in InConcert, *IEEE Data Engineering Bulletin* (June 1993).
[31] R. Medina-Mora, T. Winograd, R. Flores, F. Flores, The action workflow approach to workflow management technology, *Procs. ACM 1992 Conf. on Computer-Supported Cooperative Work*, Toronto, Canada (1992).
[32] P. Grefen, S. Joosten, E. Paalvast, R. Remmerts de Vries, *Mercurius: Design and Specifications of the Next-Generation Workflow Management Environment* (Bakkenist Management Consultants, Diemen, The Netherlands, 1995).
[33] C. Mohan, G. Alonso, R. Günthör, M. Kamath, Exotica: A research perspective on workflow management systems, *IEEE Data Engineering Bulletin* 18(1) (1995).
[34] M. Moreno, C. Rolland, C. Souveyet, A generic approach to support a way-of-working definition, *Procs. 6th Int. Conf. on Advanced Information Systems Engineering*, Utrecht, The Netherlands (1994).
[35] A. Oberweis, G. Lausen, Temporal aspects in office information systems, in, G. Bracchi, D. Tsichritzis (eds.), *Office Systems: Methods and Tools* (Elsevier Science Publishers (North-Holland), 1987).
[36] Object Management Group, *The Common Object Request Broker: Architecture and Specification, Version 2.0* (Object Management group, 1995).
[37] K. Pohl, *Process-Centered Requirements Engineering* (John Wiley, 1997).
[38] T. Schäl, *Workflow Management Systems for Process Organisations, LNCS* 1096 (Springer, Berlin, Germany, 1996).
[39] H. Schuster, S. Jablonski, P. Heinl, C.Bußler, A general framework for the execution of heterogeneous programs in workflow management systems, *Procs. 1st IFCIS Int. Conf. on Cooperative Information Systems*, Brussels, Belgium (1996).

[40] *Arena Getting Started Guide* (Systems Modelling Corporation, Sewickley, PA, USA, 1994).
[41] W. Stallings, *Handbook of Computer Communications Standards* Vol. 1 (Macmillan, New York, USA, 1987).
[42] D. Tsichritzis, A. Klug, *The ANSI/X3/SPARC DBMS Framework* (AFIPS Press, 1978).
[43] *Glossary—A Workflow Management Coalition Specification* (Workflow Management Coalition, November 1994).
[44] Workflow Management Coalition Workflow Standard—Interoperability Abstract Specification, *Doc. No. WfMC TC-1012* (Workflow Management Coalition, 1996)

Paul Grefen is currently an associate professor in the Information Systems Division of the Computer Science Department at the University of Twente in The Netherlands. He is also a member of the Center for Telematics and Information Technology at the University of Twente. From 1987 to 1992, he was a researcher in the PRISMA parallel database system project, in which he developed the transaction management and integrity control subsystems. In 1992, he received his Ph.D. from the University of Twente on the subject of integrity control in parallel database systems. He was a visiting researcher at Stanford University in 1994. Since the end of 1995, he has been involved in the WIDE ESPRIT project as a researcher and member of the technical board. The WIDE project focuses on advanced database support for workflow management systems. His current research interests include transaction management in advanced application environments and architectural design of complex information systems.

Remmert N. Remmerts de Vries is employed at Bakkenist Management Consultants, one of the larger Dutch independent management consultancies. As a partner of Bakkenist he is in charge of the consultancy group concerned with business process modelling, workflow management and document management. This consultancy group renders services from the very start until the very finish of workflow and document management systems design and development projects. During his professional career Mr. Remmerts de Vries has acquired several fields of expertise, the most important of which are project management, the design of administrative organizations, the design and simulation of business processes and the development and implementation of workflow management systems. Currently Mr. Remmerts de Vries has assignments at some Dutch government institutions and some insurance companies to select and implement workflow and document management systems. Apart from his activities as a professional consultant Mr. Remmerts de Vries is secretary of the Association of Business Engineers, an independant institution whose aim it is to professionalize the Business Engineering field of expertise.