DESIGN PRINCIPLES FOR PROCESS MODELLING IN ENTERPRISE APPLICATION INTEGRATION

PAUL JOHANNESSON AND ERIK PERJONS

Department of Computer and Systems Sciences, Stockholm University/Royal Institute of Technology,
Electrum 230, 164 40 Kista, Sweden

Abstract — There is a growing need for Enterprise Application Integration (EAI) technologies, which align the applications of an organisation to its business processes. Such technologies require an adequate methodological support so that well-structured and easily understandable models can be constructed. In this paper, such a methodological support is proposed by introducing principles for the design, validation and presentation of process models and associated data models. By applying these principles, we obtain different views of the models, and thereby facilitate the use of common models for different stakeholders, e. g. business managers, designers and operators.

Key words: Process Modelling, Design Methodology, Enterprise Application Integration, Speech Act Theory

1. BACKGROUND

Organisations today have to cope with rapidly changing customer demands, stiffening competition and innovation in production as well as information technology. Those organisations that can act swiftly in this dynamic environment will have a major advantage over their competitors. Traditionally, organisations have been functionally divided, i.e. companies have been separated into departments such as marketing, production, and service. However, the functional organisation has been shown to have a number of weaknesses. In particular, it requires a huge administration to handle issues crossing functional borders, and considerable resources are allocated to tasks that do not create value. In order to overcome the problems of a functional organisation, companies have been concentrating on business processes, i. e. the set of related activities that create value for the customers. These processes cross the functional borders of an organisation and sometimes even reach the borders of other organisations [5, 20, 26].

A focus on business processes puts the customer in the centre, and when customers demand novel products or services, the organisation can meet the requirements by adjusting its business processes accordingly. In order to handle frequent adjustments, the organisation needs flexible and integrated applications. However, in most organisations the applications have been built around the different departments or functions, i.e. marketing has built up its own IT-system with specially designed marketing applications, while production has another IT-system, containing other kinds of applications, and so on. The result has been a "stovepipe" like relationship between the function and the applications, where every function in the company is supported by its own applications, which do not communicate well across the functions in the organisation, see Fig.1.

A solution to this "stovepipe problem" may be Enterprise Software Packages, for example SAP R/3 and Baan ERP, which offer an integrated IT environment to support business processes across the functional divisions in an organisation by applying a central database to connect the different applications of the package. Still, the packages may only solve a part of the problem, as there is also a need to integrate the Enterprise Software Packages with the different legacy systems in the organisation, and to take the business process integration beyond the borders of the organisation to better collaborate with customers and other business partners [17].

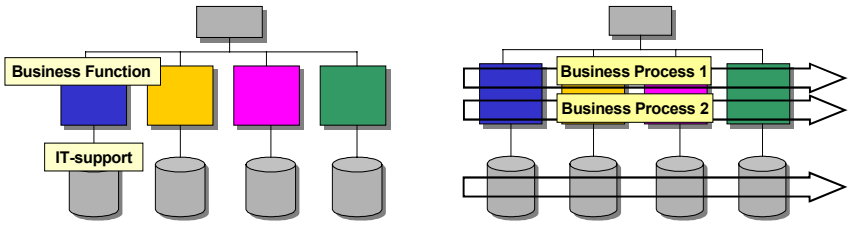

Fig. 1. The traditional function oriented structure, to the left, with the "stovepipe" like relation between business functions and IT systems. To the right the process oriented organisation, which requires an integration of the IT systems.

As a response to this need, a new breed of middleware technology has appeared, Enterprise Application Integration (EAI), which aims at integrating individual applications into a seamless whole, enabling business processes and data to speak to one another across applications, see Fig. 1. EAI makes it possible to connect Enterprise Software Packages to legacy systems and to create extended supply chains involving applications of customers, partners, and suppliers, for example through the Web. EAI also makes it possible to adopt new applications in a flexible way and thereby makes it possible to quickly create new products and services or change the existing ones. Examples of EAI technologies are Distributed Objects, e.g. Corba and DCOM, Application Servers, Message Brokers and Process Brokers, also called Process Management Systems or Process Automation Systems [17].

A Process Broker provides an integrated, graphical environment in which all process logic for connecting applications are encapsulated. The Process Broker enables users to visualise, construct, analyse, simulate and execute processes for application integration. Utilising the Process Broker technology for application integration is a complex design activity. Therefore, it requires adequate methodological support so that well-structured and easily understandable models can be produced. The purpose of this paper is to contribute to such methodological support by introducing a number of principles for the design, validation, and presentation of process models aligning the applications of an organisation to its business processes. The work presented in this paper is a result of a joint project between the Royal Institute of Technology and Viewlocity, and aims at further developing technology, methods and languages for application and process integration [19, 26].

The remainder of the paper is organised as follows. Section 2 provides a brief overview of different architectures for application integration, in particular the Process Broker architecture. The section also discusses characteristics and problems of application integration. Section 3 describes related research about process modelling languages, and in Section 4 we describe a process modelling language, BML (Business Model Language), which is used in the remainder of the paper. A classification of messages and processes, which is the base of the proposed design principles, is described in Section 5. In Section 6, we present our design principles with modelling examples and in Section 7 we discusses the need of creating an internal storage with redundant data when using a Process Broker. Finally, in Section 8, we summarise the paper and give suggestions for further research.

2. ARCHITECTURES AND PROBLEMS

2.1. Architectures for Application Integration

Integration of applications can be supported by many different architectures. One architecture for integrating applications is the point-to-point solution where every application is directly connected to every other application, see Fig. 2. This solution could work for a small number of applications, but as the number of applications increases, the number of connections quickly becomes overwhelming. The Message Broker architecture reduces this complexity, see Fig. 2. The main idea is to reduce the number of interfaces by introducing a central Message Broker and thereby make it easier to support the interfaces. If one of the applications changes format, only one connection has to be changed: the one to the Message Broker, that has tools to facilitate the format conversions [17, 27].

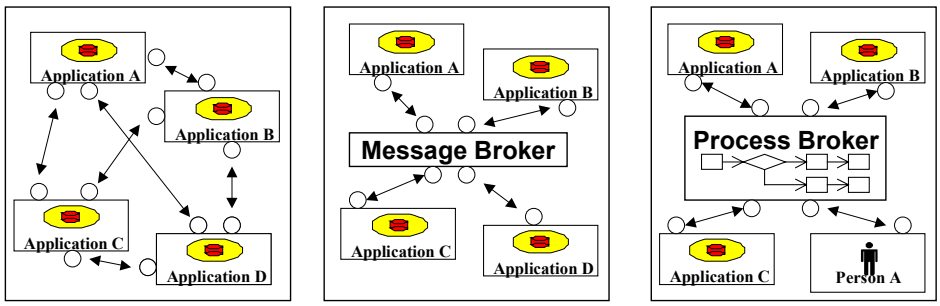

Fig. 2. The point-to-point strategy to integrate applications, to the left. In the middle the Message Broker architecture, which reduces the number of interfaces. To the right the Process Broker, which collects all process logic in the Broker.

The Process Broker, see Fig. 2, is an extension of the Message Broker. In addition to handling format conversions, the Process Broker also encapsulates the process logic for connecting applications. When all process logic resides in one place, it becomes possible to study, analyse, simulate, and change the processes using a graphical interface [4, 13, 17, 26]. This visualisation reduces the complexity and enables different categories of people to take part in the process design.

2.2. Problems in Application Integration

The Process Broker technology requires methodological support so that designers can construct models that align the applications to the business processes. Experiences of a real world modelling study at a European telecommunication company demonstrated several problems when modelling application integration. The telecom company intended to handle an integration of applications by means of a Process Broker, which aimed at facilitating the complex interaction between administrative and technical applications.

The problems in modelling application integration that we identified can be summarised as follows:

Unstructured and complex process models. Application integration often results in highly unstructured and complex models. One reason for this is that exception handling, which describes what to do when applications and human users do not respond in an expected way, makes up a large part of an application integration specification and thereby easily obscures the main business logic. Furthermore, there is often extensive communication between the Process Broker and different applications, which also tends to conceal the essential business logic.

Complex data models. Not only the process models, but also the data models can easily become complex. In particular, the mix of different types of data makes it confusing for the designers. For example, some of the data sent between the Process Broker and the external actors are central for the business processes, while other are of a more technical character, required for the proper behaviour of the external applications.

Redundancy. The Process Broker does not maintain control over external applications, which means that these applications can be updated without the Broker being notified. As a consequence, it is often desirable to maintain redundant information that duplicates parts of the information in the external applications. This redundant information enables the Process Broker to maintain a complete, correct, and easily available record of its interactions with customers. However, this duplication of information requires mechanisms for handling possible inconsistencies.

Incomplete models. Since models for application integration tend to become large and complex, there is a risk that designers overlook parts of the models that are needed to maintain completeness and consistency.

Problems:	Section 4: Choice of Language: BML	Section 5.2: Process Classifi-cation	Section 6.1: View Guidelines	Section 6.2: Complete-ness Guidelines	Section 7: Redundant Storage
Complex process models		X	X		
Complex data models			X		
Redundancy		X			X
Incomplete models				X	
Communication among stakeholders	X		X		

Table 1. The table shows in which sections of the paper the identified problems are addressed.

Communication among stakeholders. It is possible to distinguish among several kinds of stakeholders: domain experts such as business managers, owners of external applications, business and technical designers, and operators that handle the day-to-day operations. Different stakeholders require different views of the system, while at the same time they need to be able to communicate with each other using common models and languages.

In the rest of the paper, we try to address these problems by proposing a set of design principles for application integration. We also introduce a process language called BML and argue that it facilitates communication between stakeholders. Table 1 summarises how the problems identified are addressed.

3. RELATED RESEARCH

In the beginning of the 90's process orientation became one of the most important trends in management practice as well as research. Authors such as Hammer and Davenport [5, 12] advocated a radical change from a functional perspective to a process focussed perspective in order to improve customer satisfaction, shorten lead times, increase productivity, and handle technological development. Initially, process orientation achieved most attention in the manufacturing discipline, but in recent years it has also gained prominence in the information systems community [10].

A number of languages and methodologies for process specification and design have been proposed. A distinction can be drawn between *activity oriented* and *communication oriented* process languages. An activity oriented process language, e.g. UML activity diagrams or EPC [22] is intended to be used for handling arbitrary processes including material processes and involving physical actions. Therefore, the activity oriented language diagrams usually represent a mix of automated and manual actions. A communication oriented language, on the other hand, focuses on communicative processes describing the interaction between people and systems in terms of sending and receiving messages, which provides an opportunity to support the communication by means of information technology. Communication oriented languages have been heavily influenced by speech act theory [21]. One of the first systems based on a communication and speech act oriented approach was the Coordinator, developed by Winograd and Flores [25] which supported the communication process in the work place. The idea of applying a speech act based approach to information systems analysis and design was also employed by the SAMPO (Speech-Act-based office Modelling aPprOach) project in the middle of the 80's [2]. These ideas were further developed in recent work on Business Action Theory [11, 16] and in the DEMO (Dynamic Essential Modelling of Organisations) approach [6, 7]. We believe that a communication oriented approach is particularly suitable for application integration and Process Brokers, as application integration basically consists of the interchange of messages between applications and people.

A technology related to the Process Broker is the Workflow Management System [15]. The first generation of Workflow systems, during the 80's and the early 90's, was supporting communication between people, concentrating on document routing. The next generation of Workflow systems put the business processes in focus. By also involving automatic actors, the automation of the processes

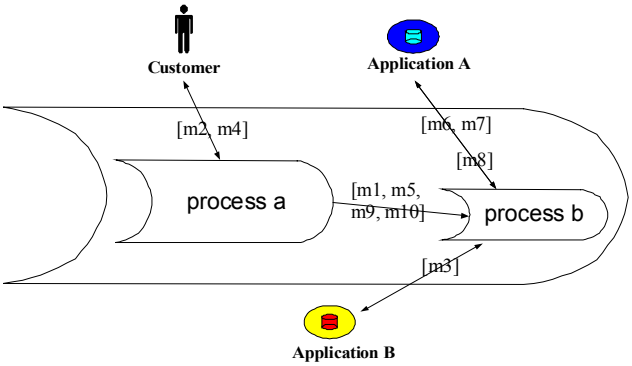

Fig. 3. The static diagram in BML visualises the structure of the processes in a system.

could be facilitated further [18]. The next step for the Workflow systems should be to implement enterprise wide workflow solutions and provide tools for managing the processes themselves. This process management includes process modelling, process reengineering as well as process implementation and automation [9, 26]. The Process Broker can be seen as this next step to process management, by providing modelling and simulation capabilities, but the Process Broker also enables rapid modifications of the business processes thanks to its flexible way of handling application integration.

In the marketplace Message Broker vendors provide functionality that enable applications to interoperate with a minimum of custom coding. Some of the major products here are Viewlocity's AMTrix and TradeSync Integration Broker [24], IBM's MQSeries Integrator [14], and Entire Broker from Software AG [23]. Several of the Message Broker vendors are adding process modelling and simulation capabilities to their products, thereby moving into the Process Broker market. Some of the major products in this market are: Viewlocity's TradeSync Process Manager [26], Extricity Software's AllianceSeries [8], Vitria Technology's BusinessWare [1], and HP's Changeengine [13].

4. BML – A LANGUAGE FOR APPLICATION INTEGRATION

To visualise the application integration processes there is a need for a process description language. This section briefly introduces such a language, BML (Business Model Language), which is developed by Viewlocity [26]. The language has similarities to SDL (Specification and Description Language) [3], but is more adapted to application integration. BML is a communication oriented process language, see Section 3, which means that it focuses on describing interactions between systems through the sending and receiving of messages. This makes the language suitable for application integration and Process Brokers. Another important advantage of BML is that the language can be used for business specification and design as well as in the execution of systems. This means that the same language can be used in different phases of a system's life cycle: in feasibility analysis, in requirement specification, in the design and implementation phases, and even in the operation phase. This enables different categories of stakeholders to use the same language for different purposes. The language can also be used directly as an implementation language and to some extent replace ordinary programming languages. Further advantages of using BML are that it is possible to describe and partition the interaction and interfaces between processes that work concurrently. Concurrency is common in application integration, e.g. when several applications are to be updated in parallel. The possibility of partitioning in BML reduces the complexity of handling large systems, through creating manageable and understandable parts with limited dependencies.

BML can be used to describe the structure as well as the behaviour of a system by using two kinds of graphical diagrams. The structure of the system is visualised by a static diagram, see Fig. 3, which describes the processes in a static mode. The static diagram describes the messages sent between the processes and between the processes and the environment, i. e. the external applications and people. The

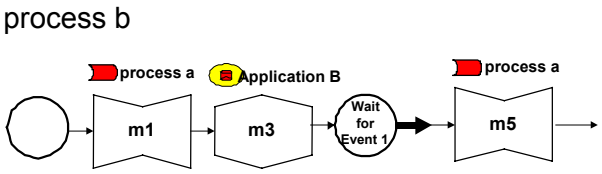

Fig. 4. The process diagram in BML visualises the order in which the messages shall be sent and received in a process. Note that the figure only shows the beginning of *process b.*

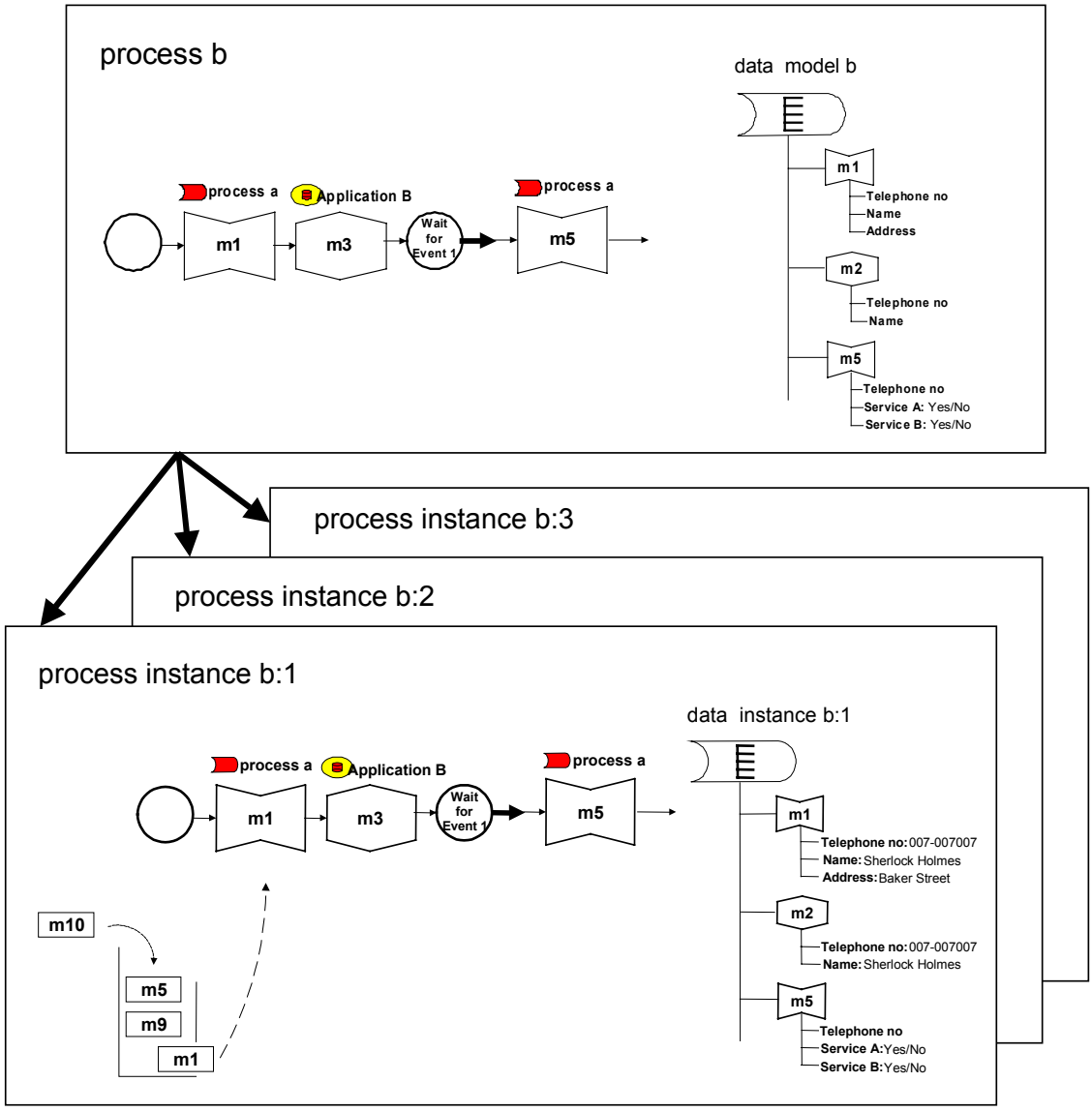

Fig. 5. The process diagram and its associated data model at the top. At the bottom the process instances with the input queues and the associated data instances.

dynamic behaviour of a system is described by using process diagrams, which visualise the control flow of the processes, i.e. in which order the messages shall be sent and received, see Fig. 4.

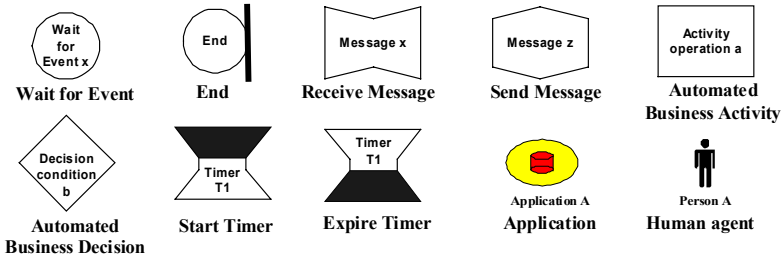

Fig. 6. Symbols used in BML.

For each process diagram there is a number of process instances. A process diagram can be seen as a template for the process instances, which are created during runtime, see Fig. 5. The process instances are executed independently of each other, but can communicate by sending and receiving messages asynchronously. Each instance has an input queue, see Fig. 5 (bottom, left), where received messages are stored. A process instance can either be waiting in a stable state (represented by a circle in BML) or perform a transition from one state to another. A transition is initiated when a message in the input queue is chosen and consumed.

Following the example in Fig. 5 (bottom), the process instance *b:1* starts in a Start State (circle without a name). Only the message *m1* can initiate a transition. The message *m1* is first in the queue and is therefore consumed, and the process instance performs a transition to the state *Wait for Event 1*. During the transition a message *m3* is sent to *Application B*. Thereafter the message *m9* is first in the queue. Since only message *m5* can initiate a further transition from *Wait for Event 1*, the message *m9* is discarded or put back at the end of the queue (depending on the implementation). The next message in the queue is then *m5*, which can initiate a transition from *Wait for Event 1* to some other Wait for Event state (not specified in the example). During the transition, data can be manipulated, decisions can be made, new process instances can be created and messages can be sent to other process instances or to the process instance itself.

The main BML symbols are the following, see also Fig. 6:

Wait for Event, Start State. The process instance is waiting in the Wait for Event state until a message is received or a timer has expired. A Wait for Event symbol without a name is the Start State.

End. Describes the end of the flow of the process instance.

Receive Message. Describes the consumption of a message from the input queue.

Send Message. Describes the sending of a message.

Automated Business Activity. Describes operations that will be performed on the data instance.

Automated Business Decision. The control flow is dynamically changed depending on different business rules.

Start Timer, Expire Timer. In application integration, the notion of time is important and timers occur frequently to obtain delays. When a timer is started it will be provided with a timeout value. The start is represented by an hourglass “full of time”, and the timeout by an hourglass “out of time”.

Application, Human actor. Both are symbols of external actors.

An important component of BML is the data model. There is one data model associated to each process diagram, see Fig. 5 (top, right). The data model describes the structure, type and meaning for the data in the Receive and Send Messages. It can be seen as the template or meta data for the data instances, which are instantiated with actual values during execution. There is one data instance connected to every process instance, see Fig. 5 (bottom, right).

Following the example in Fig. 5 (bottom, right) the message *m1* contains the values: *007-007007*, *Sherlock Holmes* and *Baker Street*. When message *m1* is received the data fields for message *m2* are

immediately updated with the values *007-007007* and *Sherlock Holmes*. Note that before values can be put in a Send Message it has to be received from a Receive Message in the process. The data fields in message *m5* are updated when *m5* is received.

It is also possible to make calculations based on data from a Receive Message and thereby create so called "derived data", which is then used as data in a Send Message.

Every process instance has a process instance key, which uniquely identifies every instance. The process instance key can either be some data from the first received message or be created automatically by the system when the process instance is initiated. The process instance key makes it possible to find the right process instance for a message. Following the example in Fig. 5 the process instance key could be the telephone no: *007-007007*. This means that every received message with the process instance key: *007-007007* is sent to the same process instance.

Other kinds of data in BML are the business parameters and timer values, which are used to configure across processes and applications. The central control and management of the Process Brokers makes this possible.

Business parameters are data, usually predefined, which are used by the Process Broker to automatically determine whether a process instance shall choose one path of execution or another (see the BML symbol Automated Business Decision in Fig. 6). For example, in an automated decision an order's sum could be examined. If the order exceeds a certain predefined sum, i.e. exceeds the business parameter, the order instance is routed along one execution path, which could notify a person, and otherwise it is routed along another path, which could instead update an application. The business parameters, e.g. the predefined sum, can easily be changed by business managers or another users. It is also possible to have business parameters that impact several processes.

Timers (BML symbols Start Timer and Expire Timer in Fig 6) are used to manage delays. For example, if an application has not answered a question at a certain time, i.e. the predefined timer value, a person could be notified. Timer values can be seen as a kind of business parameters.

5. CLASSIFICATIONS OF MESSAGES AND PROCESSES

5.1. A classification of Messages

In this section, we introduce a classification of messages as a basis for formulating the design principles in Section 6. The classification is based on speech act theory. However, it is our experience that users often find it difficult to classify messages directly according to the basic speech act types. Therefore, we introduce a set of message types that are more closely related to the messages typically found in business processes.

The study of speech acts has been an active research area in analytical philosophy since World War II, and the most influential approach to date is speech act theory as developed by John Searle [21]. Searle proposes a taxonomy for speech acts consisting of five classes: assertives, commissives, directives, declaratives, and expressives. These are also called the illocutionary points of a speech act. An assertive is a speech act, the purpose of which is to convey information about some state of affairs of the world from one agent, the speaker, to another, the hearer. An example is "It is raining". A commissive is a speech act, the purpose of which is to commit the speaker to carry out some action or to bring about some state of affairs. An example is "I promise to be at home before nine o'clock". A directive is a speech act, where the speaker requests the hearer to carry out some action or to bring about some state of affairs. An example is "Please bring me the salt". A declarative is a speech act, where the speaker brings about some state of affairs by the mere performance of the speech act. An example is "I hereby pronounce you husband and wife". An expressive is a speech act, the purpose of which is to express the speaker's attitude about some state of affairs. An example is "I like coffee".

Based on Searle's classification of speech acts, we list message types below that frequently occur in application integration. The message types are requests for information and services and the responses to these requests. We also identify messages for reserving, booking, and releasing resources. The difference between reserving and booking is that reserving is a preliminary stage to booking. A reservation could either be followed by a booking or a cancellation of the reserved resource. The distinction is important if the system automatically should cancel reserved resources that have not been booked after a certain time. For example, a person can reserve several telephone numbers for a certain time, so that he or she can

choose to book one or more numbers from them. The reserved numbers that are not booked after a time limit are automatically released so that other people can reserve or book the numbers.

The message types are the following:

Information request. An information request is a directive speech act in which the sender asks the receiver for a piece of information. *Example: What is the telephone number to the help desk?*

Service request. A service request is a directive speech act in which the sender asks the receiver to carry out a service. Typical examples of services are to deliver a product, get an authorisation, and booking a resource. In contrast to an information request, a service request does not ask for information about a state of affairs – it requires a state of affairs to be changed. *Example: Provide me with a new telephone subscription.*

Reservation request. A reservation request is a special service request in which the sender asks the receiver to reserve a resource for a period of time, meaning that the resource cannot be reserved or booked by anyone else during this period of time. *Example: Reserve five different telephone numbers (which the customer can choose from).*

Booking request. A booking request is a special service request in which the sender asks the receiver to make a resource available for the sender. *Example: Book the telephone number that the customer has chosen.*

Information confirmation. An information confirmation is an assertive speech act in which the sender, in response to an information request, provides the receiver with a piece of information. *Example: The telephone number to the help desk is 07-707070.*

Service confirmation. A service confirmation is an assertive speech act in which the sender, in response to a service request, informs the receiver that the required service has been carried out. *Example: You have been provided with a new telephone subscription.*

Reservation confirmation. A reservation confirmation is a special service confirmation in which the sender, in response to a reservation request, informs the receiver that the required reservation has been made. *Example: The telephone number is reserved (until the customer has chosen to book the number or a certain time limit has passed).*

Booking confirmation. A booking confirmation is a special service confirmation in which the sender, in response to a booking request, informs the receiver that the required booking has been made. *Example: The telephone number is booked.*

Service promise. A service promise is a commissive speech act in which the sender, in response to a service request, commits itself to carry out the required service. *Example: The delivery department promises to send the ordered telephone.*

Notification. A notification is an assertive speech act in which the sender informs the receiver about the changes of some state of affairs. *Example: The customer has started to use the subscription.*

Cancel reservation. A cancel reservation is a directive speech act in which the sender asks the receiver to cancel a previous reservation. *Example: Release a reserved number.*

Cancel booking. A cancel booking is a directive speech act in which the sender asks the receiver to cancel a previous booking. *Example: Release a booked number.*

5.2. A Classification of Processes

In this section, we introduce a classification of processes. The purpose is to support the designer in building well-structured and easily understandable application integration models. The classification identifies types of processes that are largely independent of each other and which can be combined with clear and simple interfaces. This makes it possible to partition a system of processes into manageable and understandable parts. The classification is a result of our experiences from a number of case studies.

A starting point of the classification is the customer, an actor for whom a process is to create value. By emphasising the customer, we can distinguish between customer oriented processes that directly interact with the customer, processes that support the customer oriented processes in various ways, and processes that manage more technical and informational aspects.

The classification includes the following types of processes:

Customer process. A customer process focuses on the interaction with the customer. A customer process may contain messages to or from a customer or another process type, but not to and from external actors, i.e. applications or people (except the customer). The purpose of a customer process is to show the business logic from the customer's point of view.

Interface process. An interface process handles the interaction with the external applications or people (except the customer). An interface process may contain messages to and from all other types of processes as well as to and from external applications and people. An interface process interacts with exactly one external application or person. The purpose of the interface processes is to insulate the interfaces of external applications from the main business logic. For example, when the format of messages sent from an external application changes, only the data model in the interface process has to be modified while the other processes can be left untouched. There are two subtypes of interface processes:

Request process. A request process handles information or service requests from other processes.

Release process. A release process handles cancel reservations or cancel bookings from other processes.

Synchronisation process. A synchronisation process synchronises a number of interface processes. It may contain messages to and from different types of processes, but not from external applications and people. The purpose of a synchronisation process is to encapsulate a piece of business logic – typically a synchronisation process takes care of a request from a customer process by invoking and synchronising a number of interface processes.

Maintenance process. A maintenance process handles the internal storage of information that duplicates the information in external applications (see Redundancy in Section 2.2 and Section 7). There are two subtypes of maintenance processes:

Update process. An update process takes care of a notification from a customer or synchronisation process and stores the information carried by the notification.

Consistency process. A consistency process is a process that checks whether there is any inconsistency between internally stored information and information in external applications. A consistency process also takes appropriate action when an inconsistency is detected.

A typical structure of an application integration model using the process classification is shown in Fig. 12. The customer process contains the main business logic with respect to the customer. It interacts with synchronisation processes and interface processes in order to take care of the customer's requests. It also sends notifications about the customer interaction to maintenance processes, not shown in Fig. 12. We believe that this structure supports flexibility, stability, and understandability by separating main business logic from more technical and informational aspects. By using the design principles of Section 6, designers will automatically arrive at an application integration model with the proposed process structure.

6. DESIGN PRINCIPLES

In this section, we introduce a number of design principles in the form of guidelines for the design, validation, and presentation of application integration models. These guidelines are divided into two groups. The first group consists of guidelines to obtain different views of models, while the second group consists of guidelines to check the completeness of process diagrams.

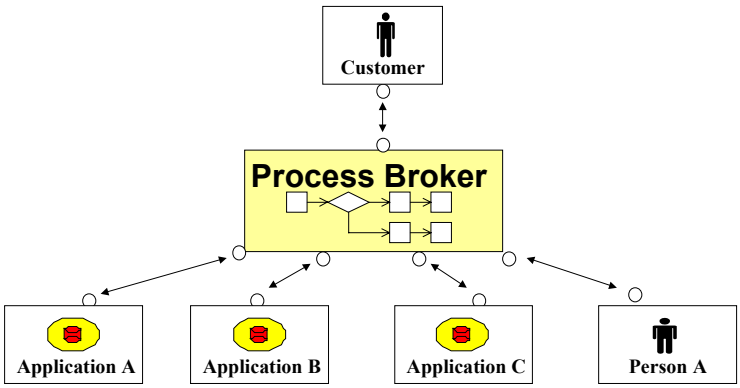

Fig. 7. The customer communicates with the applications and other people only through the Process Broker.

The main idea behind the guidelines in the first group, the view guidelines, is to obtain a series of views of processes starting with a customer oriented view on the business level. The view guidelines assume that the Process Broker is a mediator between the customer, i.e. the one for whom value is to be created, and a set of applications and people, see Fig. 7. Note that the customer does not communicate directly with the applications and other people, but only through the central Process Broker.

The succeeding views add more and more details moving from a business perspective to a more technical perspective. Each view is an extension of the previous one, either through adding subprocesses or through introducing new components into the existing diagrams. Note that the Process Broker contains all the modelled processes in the example, i. e. the Process Broker is the internal system, while the external system is the one represented by the customer, the applications and other people.

The views can be used for presentation purposes to show or hide parts of process diagrams, but also to help the designer to construct these diagrams. Therefore, the views can be seen as steps in a method for designing process diagrams.

The purpose of the guidelines in the second group, the completeness guidelines, is to support the designer in creating processes that include complete discourse structures and not only fragments. In particular, the completeness guidelines can be used to ensure that requests are always handled in an appropriate way, and that outstanding bookings and reservations are taken care of in cases of exception.

6.1. View Guidelines

The views are illustrated by means of a telephony case, in which a customer wants to order a telephone subscription. The order is updated in one application, Application A, but if the order exceeds $ 500, a manager first has to accept it.

View 1. Customer interaction. This view models the interactions between the Process Broker and the customer, i.e. the messages exchanged by the customer and the Broker as well as the flow of control. In this view, there is only one process diagram.

The first three tasks of the designer in this view are to clarify how the process is initiated, what messages the customer sends to the Process Broker, and what messages the Process Broker sends to the customer. Based on this information, the designer constructs a static diagram, see Fig. 8. The corresponding process diagram is shown in Fig. 9 (left). The process diagram shows that a customer sends a message, *Order*, to the *Customer process* that returns an answer, *Order answer*, which states whether the order is approved or not.

It should be noted here that the data model for view 1 (and 2) contains the central business information for the application integration, see Fig. 9 (right). When the data model is extended in the following views (3-6) it means that only more technical and exception information is added.

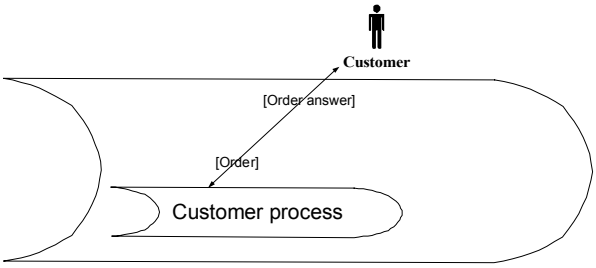

Fig. 8. The static diagram in view 1.

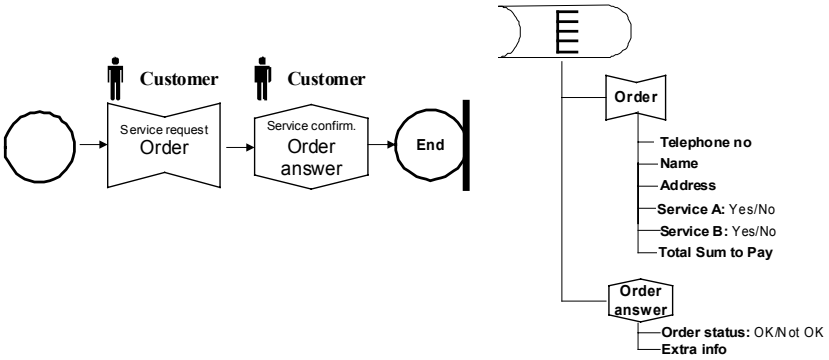

Fig. 9. The Customer process, to the left, with the data model, to the right (view 1).

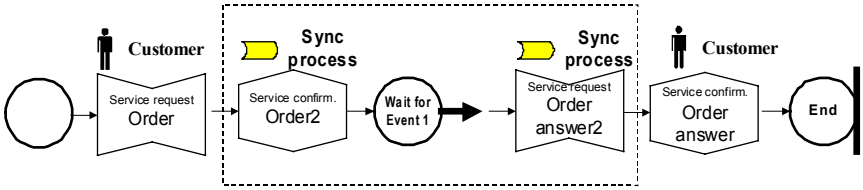

Fig. 10. The Customer process (view 2).

View 2. Broker requests. This view is an extension of view 1, which describes how the Process Broker produces the messages it sends to the customer. For each Send Message symbol from the Broker in view 1, one Send Message symbol, one Wait for Event symbol and one Receive Message symbol is added.

The designer must determine how the messages sent by the Process Broker to the customer are to be produced, which means that Send Message symbols in view 1 should be identified and analysed. Before the Send Message symbol *Order answer* in Fig. 9, one Send Message symbol, *Order 2*, one Wait for Event symbol and one Receive Message symbol, *Order answer 2*, are added, see symbols surrounded by the dotted box in Fig. 10. (In BML every Receive Message symbol must be preceded by a state symbol, i.e. Start State symbol or Wait for Event symbol, as the Broker may need to wait for a message to arrive.) The Send Message and Receive Message symbols in the dotted box represent the messages sent to and received from a new subprocess, *Sync process*. (Note that the dotted boxes are not part of the BML notation; they are used in the examples to help the reader identify the extensions for every new view introduced.) In view 2, the applications to be integrated are still not visible. They will become visible in the next view, where the introduced subprocesses are modelled.

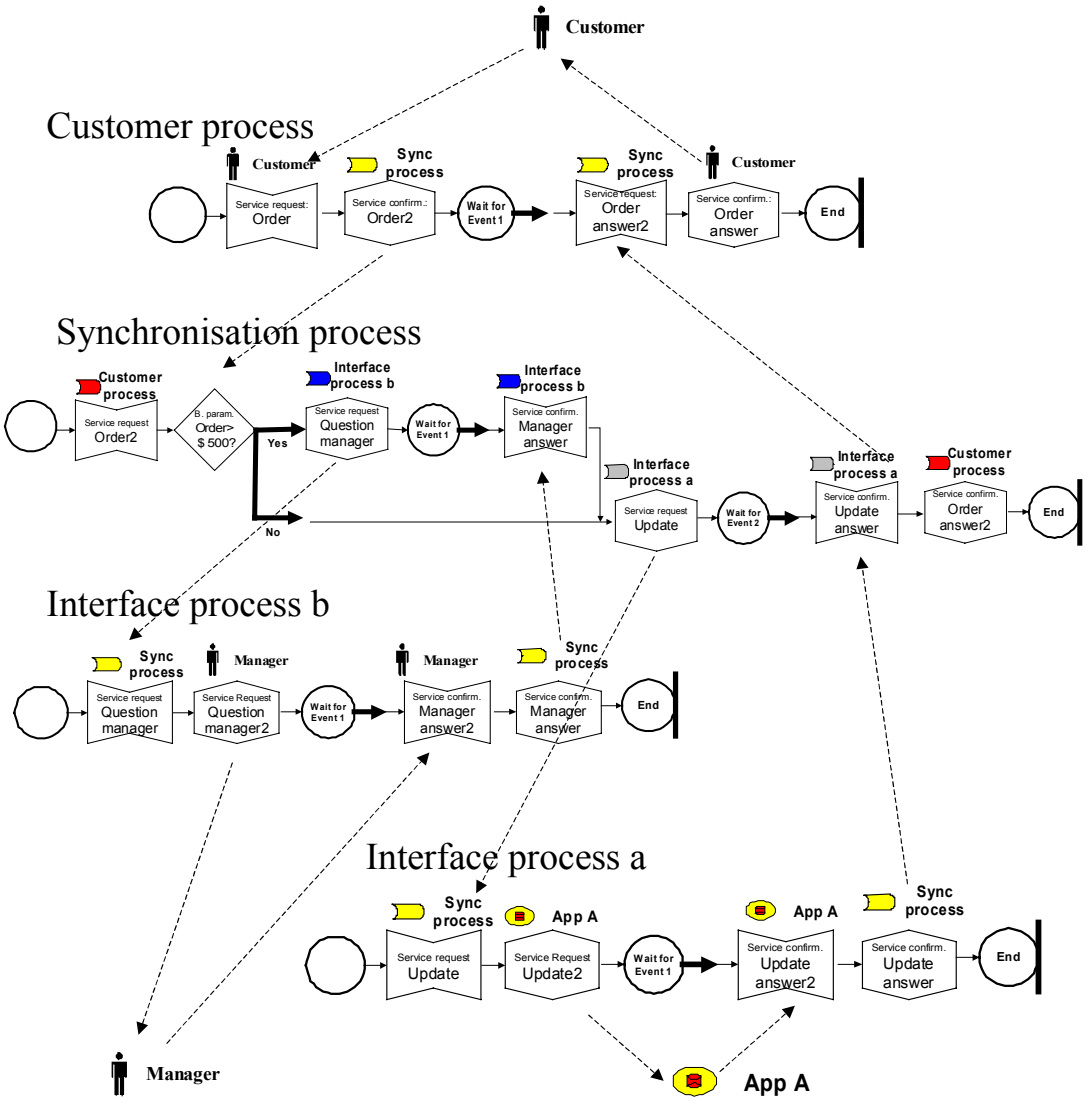

Fig. 11. The process diagrams in view 3.

View 3. External system interaction. Each subprocess introduced in view 2 is specified here. Only information and service requests and the responses to these are included in this view.

This view models the interaction with the applications to be integrated. In the simplest case, a subprocess is an interface process, i.e. it communicates with exactly one application or person, see Section 5.2. In Fig. 11 the two processes at the bottom are interface processes.

In some cases, it is convenient to introduce two or more levels of subprocesses. If the subprocess to be specified requires interaction with several applications, the designer should first construct a synchronisation process, see Section 5.2. The synchronisation process, in the middle in Fig. 11, invokes its own subprocesses and synchronises these. The relations between the different processes in the telephony case are shown in the static diagram in Fig. 12.

Every process diagram in view 3 also has a data model. Note that the *Synchronisation process* in Fig. 11 makes use of a business parameter, see Section 4. The second object in the process is an Automatic Business Decision and if the order exceed $ 500, a message is sent to a manager, who has to

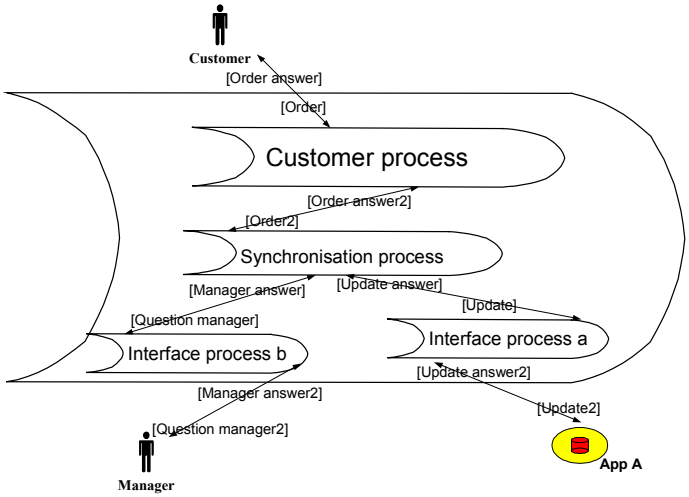

Fig. 12. The static diagram in view 3.

approve or disapprove the order. The business parameter value, i. e. $ 500, can easily be changed by a business manager or another user.

View 4. Exception handling. This view specifies the exception handling. For each Receive Message symbol, whose sender is an external application or a human actor, a Start Timer and an Expire Timer are added, see dotted boxes in Fig. 13 (top), as well as the behaviour when the exception is raised.

Views 1 – 3 specify only the normal course of events. In view 4, the designer specifies how to handle exceptions, i.e. situations where an external application or a human actor has not replied to a request within a pre-specified time limit. This means that the designer first has to extend all interface process diagrams by adding Start Timer and Expire Timer symbols, as well as the behaviour after a timer has expired. Furthermore, the designer may have to extend the process diagrams at a higher level, i.e. the synchronisation or the customer processes, in order to describe how to handle the exceptions.

The data model is extended with data that handle the exceptions. In this view also timer values (see Section 4) have to be determined. Fig. 13 (bottom) shows the data model for *Interface process a*.

In this paper we have only taken into consideration time-based exceptions. Other types of exceptions, for example value-based exceptions, could be shown in this view, view 4. However, this raises several problems, which are left for further research.

View 5. Resource releasing. This view adds all messages of the message types cancel reservation and cancel booking and also introduces release interface processes.

When a process cannot be completed as intended, it becomes necessary to release the resources that have been reserved or booked during the process. First, the designer has to extend the synchronisation or customer processes with Send Messages, of the message type cancel reservation or cancel booking, which send messages to release interface processes, see Section 5.2. Secondly, the designer has to create this release interface processes, see Fig. 14. Note that no message is sent back to the *Synchronisation process* in Fig. 14.

The data model is in this view extended with data that handle the resource releasing.

View 6. Notifications. This view adds all messages of the message type notification.

There are two main types of situations where notifications are required. First, when an exception has occurred, it is customary to inform an operator about this event so that he or she may take appropriate action, see Fig 15. Secondly, a notification may be sent to a maintenance process, see Section 5.2 and Section 7, which redundantly stores information about essential states of affairs.

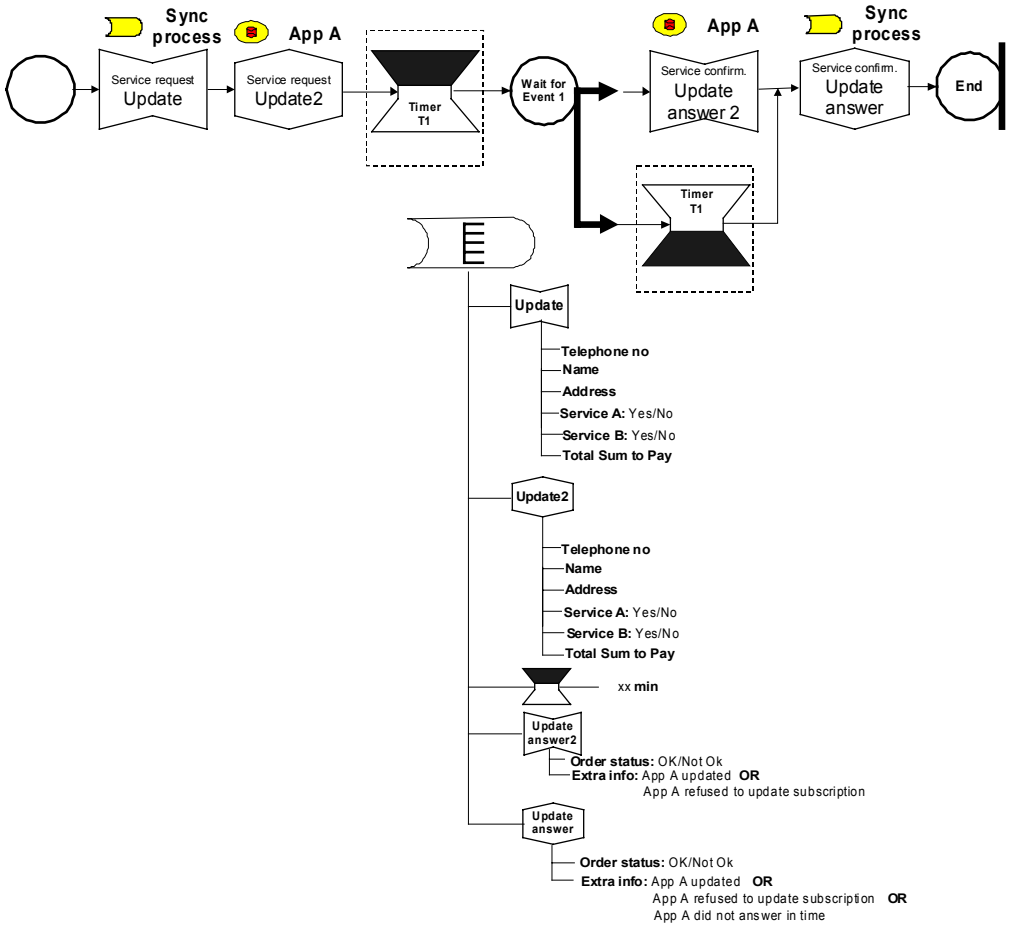

Fig.13. The Interface process a and the connected data model (view 4).

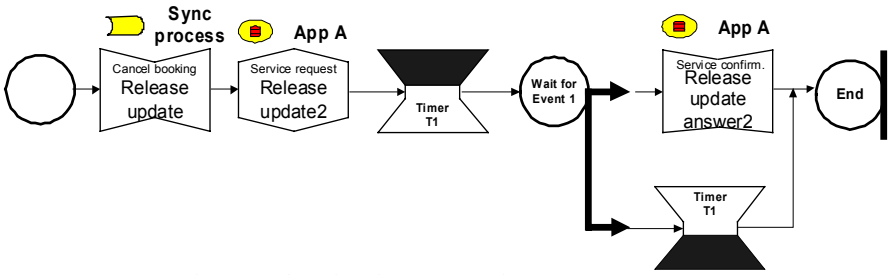

Fig.14. A release interface process (view 5).

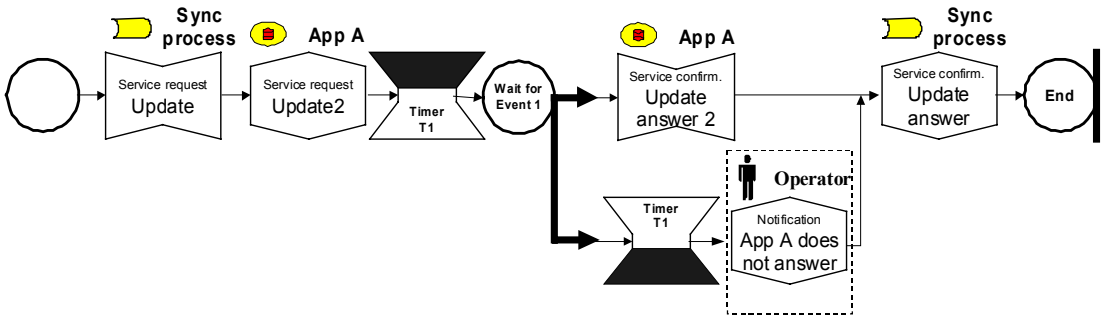

Fig.15. The Interface process a (view 6).

6.2. Completeness Guidelines

The completeness guidelines below exploit the fact that messages typically occur in certain dialogue structures. A very simple dialogue structure consists of a pair: a question followed by an answer. Another well-known dialogue structure is the conversation for basic action, introduced by Winograd and Flores in [25], which consists of the four steps: request, negotiation, fulfilment, and acknowledgement.

The following guidelines are a preliminary result of our research. Further research will result in additional guidelines.

Completeness guideline 1. In a process diagram, every service request, reservation request, booking request or information request should be followed by a corresponding service confirmation, reservation confirmation, booking confirmation or information confirmation, respectively. This pair of request and confirmation is optionally followed by a notification. This pattern (without the optional notification) can be observed in Fig 9, 10, 11 and 13. For example, in Fig. 9 the *Customer process* receives a service request from the customer and the process sends a service confirmation back to the customer.

Completeness guideline 2. In a process diagram, every reservation request should be followed by a corresponding booking request by the same actor. Fig. 16 shows a diagram following guideline 2. It should be noted that this diagram also follows guideline 1. The diagram describes a customer who wants to book a telephone number, but has to first choose a number suggested by the system. The customer can either accept the suggested number, i.e. book that number, or ask for other suggestions. The system has to reserve the number suggested to the customer, to prevent that somebody else reserves or books the number during the process.

Completeness guideline 3. In a process diagram, every reservation request should be followed by a corresponding cancel reservation. A process instance typically takes this path when some intermediate request has not been satisfied. Fig. 17 shows a diagram following guideline 3 (as well as guidelines 1 and 2). The two upper paths are the same as in Fig. 16. The two lower paths handle the exceptions, i.e. the customer cancels the reservation or the customer does not respond in a certain time. In these two latter

cases a message is also sent to a release interface process, see Section 5.2, which releases the reserved resources.

Completeness guideline 4. In a process diagram, every booking request could be followed by a corresponding cancel booking. A process instance typically takes this path when some intermediate request has not been satisfied. This is a weaker guideline than the previous three that were proposed.

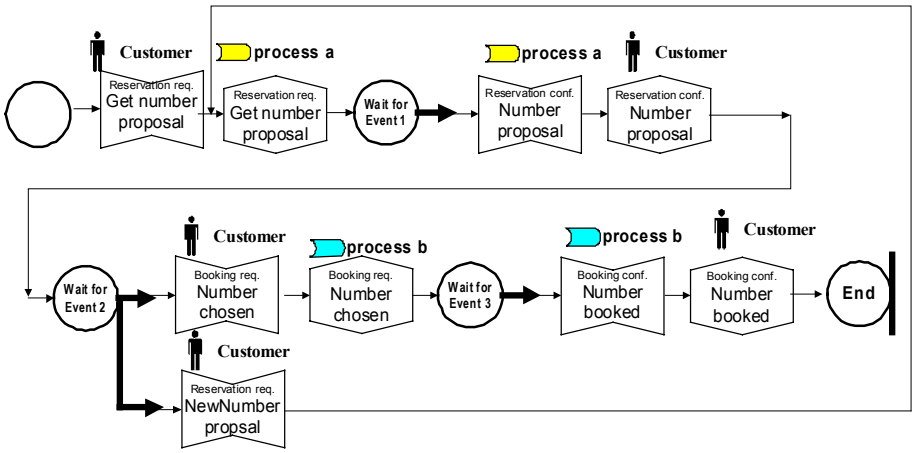

Fig.16. A customer process handling reservation and booking (view 2).

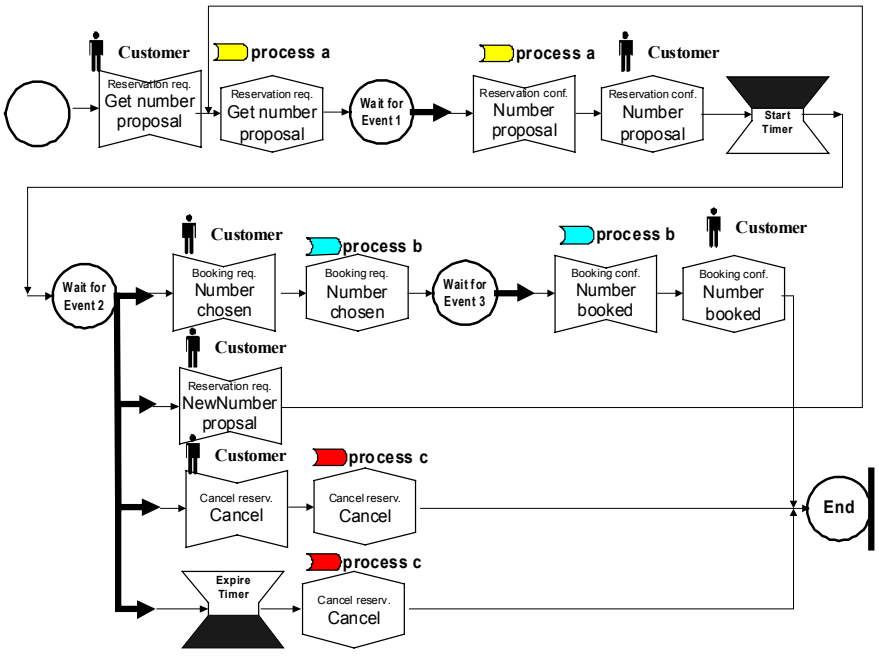

Fig.17. A customer process handling reservation, booking and cancel reservation (view 5).

7. REDUNDANT STORAGE

When using a Process Broker it is often desirable, but not necessary, to maintain redundant information in an internal or a master storage, which duplicates part of the data in the external applications. There are several reasons for this.

First, it is possible that the external application can be updated without the Process Broker being notified. For example, the data about a customer can be changed by mistake when an external application communicates with applications not connected to the Process Broker. By using an internal storage the system has complete and correct data saved in one place.

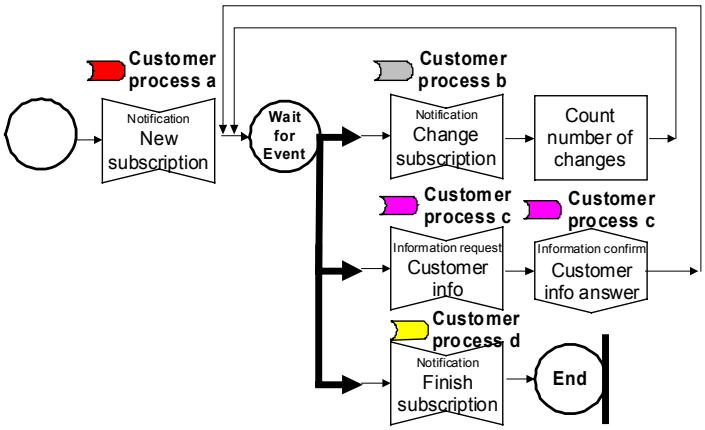

Fig.18. A maintenance process handling subscription.

Secondly, if the customer or customer service quickly requires information about, for example, an order, the system does not have to query several external applications; it only has to query the internal storage. It is not unusual that information about a customer is distributed over several applications. In that case, the internal storage is from a performance perspective, an important improvement of a system.

Thirdly, the opportunity to notify an internal storage of important events that occur in the system, makes it possible to create a data warehouse that is updated in real-time. A Process Broker can support data movement in real-time and it thereby can provide business managers with constantly new information, which can be aggregated.

The internal storage is handled by a maintenance process, see Section 5.2. The business managers and business designers first have to decide which events should notify the internal storage and then extend the customer and synchronisation processes with Send Message symbols, i.e. the messages that are sent to the maintenance process. Following this, the maintenance process has to be modelled. Fig. 18 shows a maintenance process that is handling subscription information. When a new telephone subscription is ordered, a message, *New subscription*, is sent from the customer process to the maintenance process and a new instance of that process is created. The instance is then waiting for three different messages: *Change subscription*, if the customer wants to change the subscription, *Customer info*, if the customer wants to know what he or she has ordered, or *Finish subscription*, if the customer wants to finish the subscription. The maintenance process visualises the activities, i.e. updates and queries that can be performed on the storage. Furthermore, an aggregation made on the internal storage data can be modelled. For example, the process in Fig 18 counts how many times a certain subscription is changed.

However, the redundant information in the internal storage and the external applications, requires a process for handling possible inconsistencies. This consistency process, see Section 5.2, checks whether there is any inconsistency and takes appropriate actions when any inconsistency is detected.

8. SUMMARY AND FURTHER RESEARCH

In this paper we have addressed methodological support for modelling the alignment of application integration to business processes. The main contribution is the guidelines introduced above which can be

used in several ways. First, they can be used in design. A designer could utilise the view guidelines by first constructing a process diagram according to view 1 and then gradually refining it until a set of diagrams in view 6 is obtained. Furthermore, the designer can use the completeness guidelines to guide the design of each individual process diagram. Secondly, the guidelines can be used to assist in the validation and verification of process diagrams. By checking the completeness guidelines, a designer can ensure that essential parts of discourse structures and exception handling are not omitted. Thirdly, the guidelines can be used for presentation purposes. Business oriented users can choose to see only the top view or views, while technical designers and implementers can proceed to the lower views. Even for the latter category of stakeholders, the layered views can help to understand a system by allowing to focus on an essential business perspective first and thereafter proceed to a technical perspective. Different categories of users, for example customers, business and technical designers, have the possibility of suggesting input on the right level in the modelling process. Business designers probably want to concentrate on the important parts of the business processes to clarify how they want the main business logic to work.

In section 2.2 we identified several problems when modelling application integration, which have been addressed by a number of instruments: speech act theory, process classification, design guidelines, and a communication oriented language. A major problem with process and data models describing application integration is the need to include a large amount of details, not least because of the extensive exception handling. This results in complex and cluttered diagrams. Therefore, we propose a set of view guidelines that enable users to choose on what level of details they want to study their models. Also the process classification is intended to facilitate the creation of well-structured and easily understandable process models in application integration. Furthermore, when working with complex models it is easy to forget the necessary parts of the models. Therefore, some kind of validation is desirable. The completeness guidelines are intended to help the designers to find out whether certain symbols in the models are missing. Another problem is that communication between different stakeholders can become difficult if they use different models and modelling languages. By choosing a common modelling language, BML, and making use of views when presenting the models, we believe that communication will be facilitated. Finally, when using a Process Broker there is often a need to duplicate part of the information in the external applications and store it in an internal or a master storage. However, this redundancy increases the risk of inconsistency, which must be handled. Therefore, we introduce maintenance processes that handle the internal storage and consistency processes that detect and handle inconsistencies.

We intend to follow up the work presented in this paper by further research, which goes in several directions. One is to compare designers following the suggested view guidelines with designers who are not following them. There is also a possibility to let different kinds of stakeholders to design the models. Such an empirical study could provide input to refined or additional design principles. Another direction is to find further messages types. For example, when an actor wants to reserve, book or ask for a quotation at another actor there are several arrangements the two actors can agree on in terms of time limits, payment, how to cancel a reservation/booking and legal questions. To find and define message types that describe these arrangements should make it easier to automate the different agreements. A third research direction is to find further dialog structures that produce additional completeness guidelines for validation of the models.

Acknowledgement - This work was performed as part of the NUTEK (Swedish National Board for Industrial and Technical Development) sponsored project Process Broker [19]. The authors are grateful to Jan-Owe Halldén, Mikael Nilsson, Jan Törnebohm and Christer Wåhlander at Viewlocity for their valuable suggestions and knowledge. We also thank our colleagues at the Royal Institute of Technology, especially Birger Andersson, S.J. Paheerathan**,** Prasad Jayaweera and Nasrin Shakeri for commenting on earlier versions of this paper.

REFERENCES

[1] R. Atwood. Bringing Process Automation to Bear on the Task of Business Integration. Vitria Technology. In *http://www.vitria.com/products/whitepapers/ seyboldwp.html* (1999).

[2] E. Auramäki, E. Lehtinen, K. Lyytinen. A Speech Act Based Office Modelling Approach. In *ACM Transactions on Office Information systems,* **6**(2):126-152 (1988).

[3] F. Belina, D. Hogrefe, S. Amardeo. *SDL with Applications from Protocol Specification*. Carl Hanser Verlag and Prentice Hall International, UK (1991).

[4] P. Butterworth. Automating the Business Processes of Mission-Critical Distributed Applications. Forté Software. In *http://www.forte.com/product/downloads.html* (1997).

[5] T. Davenport. *Process Innovation: Reengineering work through information technology.* Business School Press, Boston (1993).

[6] J. Dietz. Business Modeling for Business Redesign. In *Proceedings of the 27th Hawaii International Conference on System Science,* IEEE Computer Society Press, pp. 723-732 (1994).

[7] J. Dietz. Modelling Communication in Organizations. In *Linguistic Instruments in Knowledge Engineering,* R.v.d. Riet, editor, Elsevier Science Publishers, pp. 131-142 (1992).

[8] Extricity Software. Extricity AllianceSeries. In *http://www.extricity.com/products/alli_series_over.html* (2000).

[9] D. Georgakopoulos, M. Hornick, A. Sheth. An Overview of Workflow management: From Process Modeling to Workflow Automation Infrastructure. In *Distributed and Parallel Databases*, **3**(2):119-153 (1995).

[10] P. Green, M. Rosemann. An Ontological Analysis of Integrated Process Modelling. In *Proceedings of the 11th International Conference*, *CaiSE'99*, pp. 225-240, Springer-Verlag, Heidelberg (1999).

[11] G. Goldkuhl. Generic Business Frameworks and Action Modelling. In *Proceedings of Conference Language/Action Perspective'96*, Springer-Verlag (1996).

[12] M. Hammer, J. Champy. *Reengineering the Corporation. A manifesto for Business revolution.* Harper Business*,* New York (1993).

[13] Hewlett Packard Company. HP Changengine Overview. In *http://www.ice.hp.com/ cyc/af/00/101-0110.dir/aovm.pdf* (1998).

[14] IBM. MQSeries Integrator. In *http://www-4.ibm.com/software/ts/mqseries/ (*1999).

[15] S. Jablonski, C. Bussler. *Workflow Management*. Thomson, London (1996).

[16] M. Lind, G. Goldkuhl. Reconstruction of different Business Processes – A Theory and Method Driven Analysis. In *Proceedings of Conference on Language/Action Perspective '97,* Veldhoven (1997).

[17] D. Linthicum. *Enterprise Application Integration.* Addison-Wesley (2000).

[18] P. Makey, editor. *Workflow: Integrating the Enterprise.* Butler Group report, Hessle (1996).

[19] Process Broker Project. Process Broker architecture for system integration. In *http://www.dsv.su.se/~pajo/arrange/index.html (*1999).

[20] G. Riempp. *Wide Area Workflow Management: Creating Partnership for the 21st Century*. Springer-Verlag (1998).

[21] J. R. Searle. *Speech Acts – An Essay in the Philosophy of Language*. Cambridge University Press (1969).

[22] A. Sheer. *ARIS-Business Process Modelling.* Springer-Verlag, Berlin (1998).

[23] Software AG. Entire Broker. In *http://www.softwareag.com/corporat/solutions/ applintegr/default.htm#b1* (1999).

[24] Viewlocity. AMTrix. In *http://www.viewlocity.com/solutions/* (1999).

[25] T. Winograd, F. Flores. *Understanding Computers and Cognition: A New Foundation for Design*. Ablex, Norwood, N.J. (1986)

[26] C. Wåhlander, M. Nilsson, A. Skoog. *Introduction to Business Model Language & SmartSync Model Manager*. Copyright Viewlocity (1998).

[27] L. Yeamans. *Enterprise Application Integration*. NSAG Inc. In *http://www.messageq.com/EAI_survival.html (*1999).