Subject-oriented Development of Federated Systems – A Methodological Approach

Albert Fleischmann
Metasonic AG
Pfaffenhofen, Germany
albert.fleischmann@metasonic.de

Werner Schmidt
Technische Hochschule Ingolstadt
Ingolstadt, Germany
werner.schmidt@thi.de

Christian Stary
Johannes Kepler University
Linz, Austria
christian.stary@jku.at

***Abstract*—This paper describes an approach for developing federated IT systems. Those are necessary if independent enterprises want to combine their services to reach common objectives without giving up their independence. According to Conway's law, organizations produce system designs which fit their organizational structures which might be insufficient for networks. We propose a software development methodology which fits to federated organizations. The proposal is based on Subject-oriented Business Process Management (S-BPM). In S-BPM systems are specified as a set of communicating subjects allowing for both, independent and federated operation. We detail the respective development activities based on subject-oriented modeling.**

Keywords—Software Project Management, Subject-oriented BPM, Cross-Company Software Development Projects, Federated Systems, Virtual Enterprises.

I. INTRODUCTION

In our global economy enterprises cooperate around the globe in order to create services or manufacture products for customers which are also distributed all over the world. The challenge of the cooperating partners as a federation of independent systems (virtual enterprise, VE) is to establish smooth cross-enterprise communication to reach the common objectives [1]. Information and communication technologies (ICT) are essential to create a federation of independent software systems suitable to execute business processes across the involved companies. Figure 1 shows an example of an order-to-cash scenario where federated applications support a cross-company business process. A dog food store sells its products via internet. It commissions a transportation service provider to deliver the ordered products to the customer, who confirms the reception of the goods. The store deducts the money from the customer's bank account. The process steps are facilitated by several independent software applications and message exchanges (order, order confirmation, delivery notification etc.) enabled by respective communication systems.

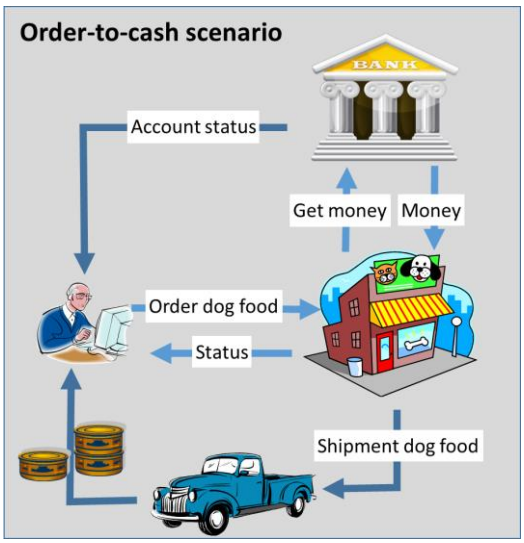

Figure 1: Order-to-cash scenario in a federation of enterprises and applications (simplified)

Developing such a mutually adjusted solution by a federation of independent enterprises requires an approach different from traditional software development projects taking a process perspective (cf. [2]). Therefore our focus is on how to implement loosely coupled systems for exchanging information between independent partners, rather than tightly coupled solutions for sharing information or other resources.

The article is structured as follows. In section II we first take a look at virtual enterprises, federations of enterprise information systems, and their peculiarities, as they form the conceptual background of the proposal. Then, software development methodology and its elements are reviewed with respect to developing federated systems. This leads to section III, containing our proposal of a software development approach for federated systems based on subject orientation. We conclude in section IV.

II. BACKGROUND

A. Recommendations for creating federated systems

When independent enterprises develop a federated system a lot of managerial and technological aspects have to be considered, particularly with respect to managing collaborative business processes. This is reflected in the following recommendations (cf. [3], [4]):

1. Start the foundation of a federation and identify members.
2. Identify and describe the business services that organizations can provide or they need from partners in service level agreements.
3. Harmonize the enactment of collaboration by coordinating the participating organizations according to defined business processes and identify the systems required for the federation.
4. Integrate the identified and implemented services/systems into the intended application.
5. Maximize the autonomy of organizations when collaborating, thereby ensuring organizations to benefit most from their own business objectives.
6. Represent the partnerships between collaborating organizations when collaborating, and update changes in partnership.
7. Guarantee the business privacy of organizations in the course of collaboration.
8. Allow partners and other third parties to monitor, measure, and oversee the execution of business processes.

B. Federation of enterprise information systems

[1] define virtual enterprises and federations of enterprise information systems as follows: "*The Enterprise partners' Virtual Enterprise (EP VE) is the federation of partners in the community that come together to achieve the goal of a federated distributed system environment, sharing their resources, and collaborating to achieve a common goal: the Federated System VE (FS VE). The partners in the federation retain autonomy over their resources, deciding which resources (personnel, resource dollars, equipment, etc.) are sharable for achieving this goal. The results of this VE are then useable by the partners in furthering their individual systems. The FS VE is seen to be a virtual system of distributed processing components (hardware and software), which are physically implemented and managed by the partners. It is a federation of the partners' systems, where each system retains its autonomy over all processing system components and sharable data/information. Retaining autonomy means defining which data or information and software/hardware assets will participate in the federation and be accessible and usable by other systems in the federation."*

The definition shows that the focus is on sharable resources. This means when setting up a federation the VE members need to clarify ownership of the shared resources as well as access rights and the rights to change those. Such an approach often implies tight coupling of the involved enterprises and the related resources. Entities leaving a federation then cause difficulties with respect to separating involved systems (changing access rights) and sorting out ownership of information.

Alternatively, information can be exchanged between the partners by messages, implying only a loose coupling of the involved systems. In this case the partners only need to agree upon structure and meaning of the data, e.g., using XML schemes, and upon the implementation of the message exchange, e.g., by web services.

C. Software development methodology

"*A software development methodology is a collection of procedures, techniques, tools and documentation aids which help developers to implement software systems*" [5]. It may include modeling concepts, tools for model-driven architecture, integrated development environments (IDEs) etc. The so-called magic triangle (see figure 2) summarizes the various aspects of a software development methodology [6].

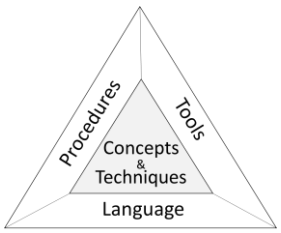

Figure 2: Magic triangle of software development methodologies

Concepts and Techniques are used to create models of the software to be implemented, and are thus significantly influencing which languages, procedures and tools are utilized. The applied concept implies the artifacts to be produced, of which the executable software system is the most important one. The Language is used to create the artifacts and tools. Procedures describe the sequence in which the activities for creating the various artifacts are executed. While languages and tools can be replaced without impacting concepts and procedures, the latter are decisively determining the shape of a software development environment.

D. Modeling concepts

Developing a federated system like the one described in section I requires modeling cross-company business processes and the entities performing activities in these processes.

1) Business process modeling

There are various approaches for specifying business process models. IT implementations of those models are called process-controlled applications [7] or workflows. The modeling approaches can be distinguished in three classes: (i) Control flow-based specifications put the focus on the activities. (ii) Object-based models mainly describe business objects and the sequence of operations to manipulate them. (iii) Communication-based models focus on the active entities in a process which exchange messages in order to coordinate their work.

By their nature the latter are promising candidates for modeling federations of systems. Business Process Model and Notation (BPMN), the currently most widely discussed modeling language, contains elements for the description of

control flows and communication in business processes. In the following we discuss its communication-oriented features.

To model communication BPMN provides so-called pools, each representing a process that can exchange messages with processes in other pools. Conversation diagrams are the means to describe this mechanism: However, they do not allow specifying the sequence in which messages are exchanged. Although the sequence can be captured by collaboration diagrams, the semantics of sending and receiving messages is not precisely defined. For instance, it remains unclear whether messages are exchanged synchronously or asynchronously. Additionally a certain message from a pool can only be received in a single activity state, but not in other states. Choreography diagrams in BPMN also define the allowed message sequence between pools. [8] describe a choreography-based tool for specifying global processes. The problem is that choreography specifications cannot contain data. As a consequence a modeler can only describe message sequences being covered by regular expressions, which is the lowest level in the Chomsky hierarchy. This fact makes it impossible to model a behavior like the following: Pool *S* sends *n* messages of a type *X* to pool *R*. After that *S* sends a message *Y* to *R*. Subsequently *S* expects *m* messages of type *A* from pool *R*, which received the *n* messages of type *X*. The reason for that is that the messages cannot be counted, because data are not allowed in BPMN choreographies.

Given these properties of BPMN this notation has significant draw backs for modeling communication, hindering the precise development of federations of systems.

2) Multi-agent systems modeling

The term agent has multiple meanings. We follow the definition given in [9]: An agent is an entity that performs a specific activity in an environment of which it is aware and that can respond to changes. A multi-agent system (MAS) is a system where several, perhaps all, of the connected entities are agents. The most important property of agents is their controlled autonomy: They independently execute their role-specific behavior, and in multi-agent systems they communicate with each other. These properties are alike those of federated systems which therefore can be considered as multi-agent systems. This means that software development methodologies for agent-oriented software (for an overview see [8]) can help developing federations of applications.

E. Procedures

Software Life Cycles (SLC) build a framework for software development procedures. All software development projects follow a series of phases. While software life cycles can be defined in many different ways, each of them comprises the following generic activities:

- Project conception or initiation
- Planning
- Execution with specification and implementation activities
- Termination

In the traditional waterfall approach these activities are performed in the sequence shown above. Other life cycle concepts propose overlapping the development steps, suggest alternatives like the V model or agile development procedures like Extreme Programming and Scrum. [10], [6] and [5] give an overview of the various approaches.

F. Work break down structure (WBS)

The work break-down structure describes the work to be done in a project in a hierarchical way. A work break-down structure element may be a product, data, service, or activity contained in the software life cycle or any combination thereof. A WBS also provides the necessary framework for detailed cost estimating and control along with guidance for schedule development and control. The top level of the WBS should identify the major phases and milestones of the project in a summative fashion. Consequently, the phases used in the top level depend on the software development methodology applied in a project. The first level can either represent the phases used in the software life cycle or the major artifacts of the system to be developed. In case the top level is SLC-oriented it might be built by requirement specification, software architecture, programming, test etc. In the case of an evolutionary life cycle there will be topics like Release 1, Release 2 etc., followed by headlines like requirement specification on the second level.

Another alternative is to use top level headlines corresponding to artifacts created by modeling activities, such as ‘create communication structure’ or ‘describe subject behavior’ (see section III.D).

The WBS is created during the planning phase of a project life cycle. During this phase the project manager works with the project team to make sure that the client's needs are addressed and the project is planned completely and approved by the client prior to any sort of production beginning on the project.

G. Organisational break down structure and software architecture

An organizational breakdown structure (OBS) complements the WBS and resource breakdown structure of a project. Project organizations can be broken down in much the same way as the work or product. The OBS is created to reflect the strategy for managing the various aspects of the project and shows the hierarchical breakdown of the management structure. Hence, the work break down structure has a significant impact on the organizational structure of the project team. The same holds for the phases of the software life cycle and the system architecture influencing the work break down structure. Conway’s law states “*organizations which design systems ... are constrained to produce designs which are copies of the communication structures of these organizations*” [11]. A variation of Conway’s law can be found in [12]. "*If the parts of an organization (e.g., teams, departments, or subdivisions) do not closely reflect the essential parts of the product, or if the relationship between organizations do not reflect the relationships between product parts, then the project will be in trouble... Therefore: Make sure the organization is compatible with the product architecture*” [12].

As we look at developing federations of systems with a federation of independent project teams, the system architecture needs to be aligned with the multiple project team structure.

III. SOFTWARE DEVELOPMENT METHODOLOGY FOR FEDERATED SYSTEMS

The software development methodology for federated systems proposed here is based on Subject-oriented Business Process Management (S-BPM) as the most straightforward enabler of communication-oriented BPM [16]. Therefore we first outline this approach the way being used in many industrial projects successfully (for examples see [13], [14], [15]). We then explain all activities and steps of the development cycle for describing and implementing a federated system using the dog food order-to-cash example. The sample case starts with having the idea to create a solution and ends with a running application.

A. Subject-oriented business process models

1) Subjects, messages, and business objects

Subject-oriented business process management includes a modeling language for describing processes as a system of independent entities which organize their work by exchanging messages. Each entity is autonomous in the sense that it decides by itself when it sends messages, receives messages and executes internal actions. These entities are called subjects and can be interpreted as roles in a process to be implemented. Each subject has its local data which can be changed by local actions or by receiving messages. These data are called the business objects of a subject. Each message has a name, similar to a name of a method in object-oriented systems, and related business objects, which are send or received with it. If a message is sent it transmits the values of the business object. If a subject accepts a message by picking it up from the input pool (see section III.A.3), the values of the incoming business object are copied into a corresponding local business object of the receiving subject.

Physical things like a can of dog food can also be a business object. In general physical business objects are accompanied by data-oriented ones like a delivery slip. Consequently, a can of dog food together with the data of a delivery slip may form a combined business object. In the model data and physical entities are considered in the same logical way. The physical aspect of a business object is considered as implementation aspect. Such an understanding allows creating a model on a logical level independent from implementation details. These are added later on.

The sequence in which messages are sent, received, or internal actions are executed, is defined by the subject behavior (for details see section II.A.4).

2) Subjects and agents

In order to execute subjects they will be assigned to agents or actors. An agent in that context is a human or non-human entity which is capable to execute actions. Details about the relationship between subjects and agents/actors can be found in [16]. This distinction allows to specify a federated system independent from a special environment. The subjects represent the members of a federation, and subjects can be assigned to another agent once the business relationship changes. Subject-oriented models are therefore independent from special members of a federation. For details about the deployment of subjects see [16] and [17] .

3) Communication structure and message exchange

A communication structure shows which subjects are involved in a process and which messages they exchange. Figure 3 depicts the communication structure of the dog food order-to-cash application in the so-called subject interaction diagram (SID). It describes the system on a logical layer. What it does not show, is sending the message “deliver dog food” that is implemented by a truck transporting dog food cans together with a delivery slip. As we will see later on the exchange of all other messages are implemented using information and communication technology, and thus considered as aspects only relevant for implementation, but not for modeling (i.e. designing).

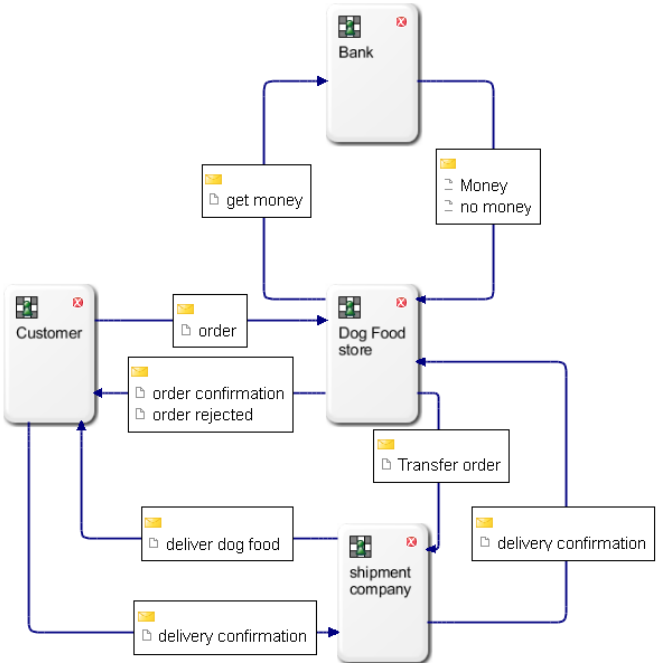

Figure 3: Communication structure of dog food order-to-cash application

The specification of the communication structure includes the business objects of each subject. It also defines which business object values are transported by which messages. The maximum number of messages which can be deposited in an input pool determines its size, independent on the size of the business objects coming with a message. In case the size limit is reached sending will be blocked until the receiving subject removes a message from its input pool. Alternatively, the incoming message replaces the most recent or message received initially in the pool, according to the chosen strategy. Details about message transfer and the related synchronization mechanisms are explained in [17].

A subject can receive certain messages by checking the input pool on their availability and removing those it finds. On removal the values of the transmitted business object are copied into a local business object.

Figure 4 depicts the input pools of the subjects in our example. Since each subject sends one message to other subjects and then waits for an answer, all input pools have a maximum size of one.

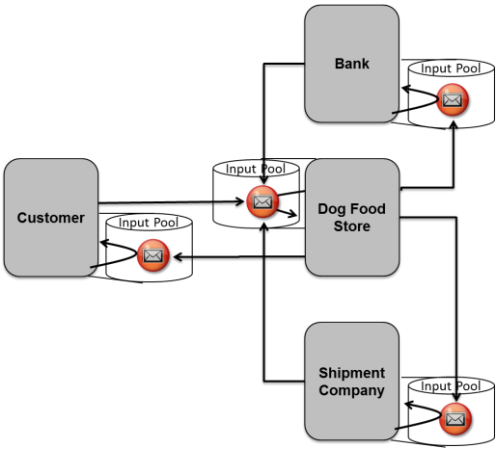

Figure 4: Message exchange via input pools

4) Subject behavior

Each subject executes send, receive and internal operations in a certain sequence. The allowed sequence is defined in a subject behavior diagram (SBD). Figure 5 shows the SBD of the subject "dog food store". In the start state "wait for…" the subject waits for the message "order" from subject "customer". After that the message "get money" is sent to the subject "bank". If the message "money" comes from the subject "bank" the message "order confirmation" is sent to the customer. Then, the internal operation "prepare order" is executed.

This function includes activities like checking the availability of the ordered goods, preparing the dog food, and all accompanying documents for shipment and updating the inventory. If the goods are not on stock the customer gets an order confirmation indicating a delay. In case of availability the message "Transfer order" is sent to the subject "shipment company". It contains data (delivery slip) and physical business objects (cans with food). According to the behavior specification the subject then waits until the delivery confirmation arrives from the shipment company and subsequently terminates processing the instance.

B. Development as a multiple-team structure

We now assume that the dog food order-to-cash scenario does not yet exist. The store wants to extend its services for the customers by offering online shopping and home delivery. In order to reach this business objective it takes the initiative to found a federation of enterprises which combine their services and develop a corresponding federation of systems.

Each federated enterprise establishes a project team, working on their parts of the solution independent from each other. This leads to a multiple-team project on the federation level [18]. As the teams belong to different, independent companies they all have their own development culture and methodology.

Since there is no single line management who can assign an overall project manager, the federation members need to agree on a project leader and the competencies related to this role. As the initiator of a federation has the most interest in the development of the federated solution it might be helpful that this company, in our case the store, recruits the leader.

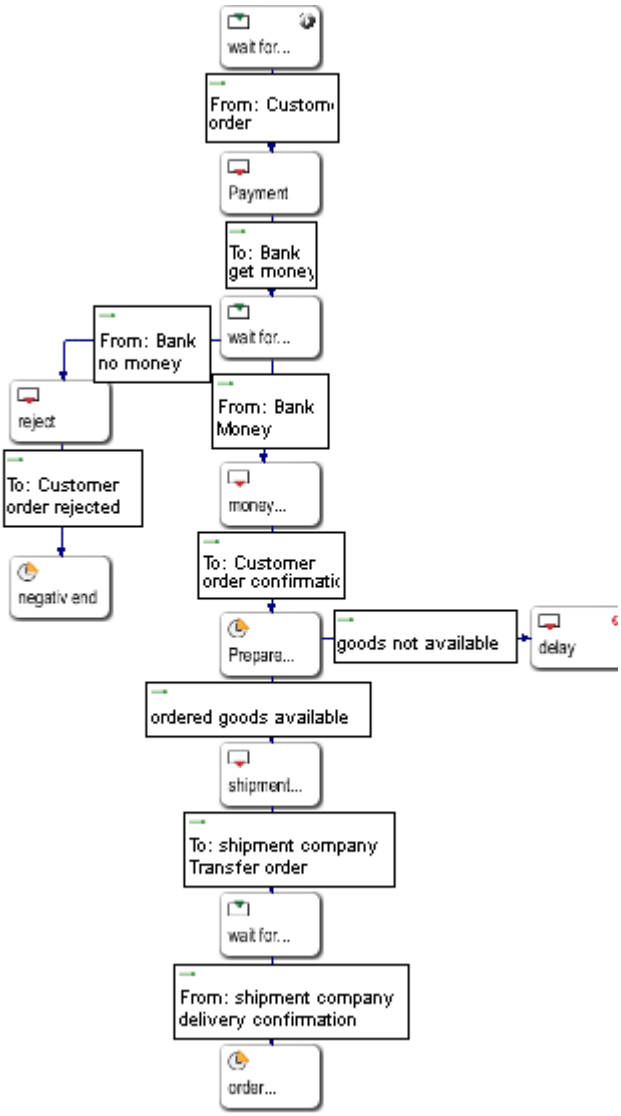

Figure 5: Behavior of the dog food store (clipped)

His or her major task is to ensure smooth communication between the independent teams, respectively their managers. The project teams needs to coordinate how the systems they are developing communicate with each other. Their major communication paths are predefined by the communication structure of the system federation. This strategy leads to a high socio-technical-congruence. Figure 6 shows the team and communication structure of the dog food order-to-cash federation.

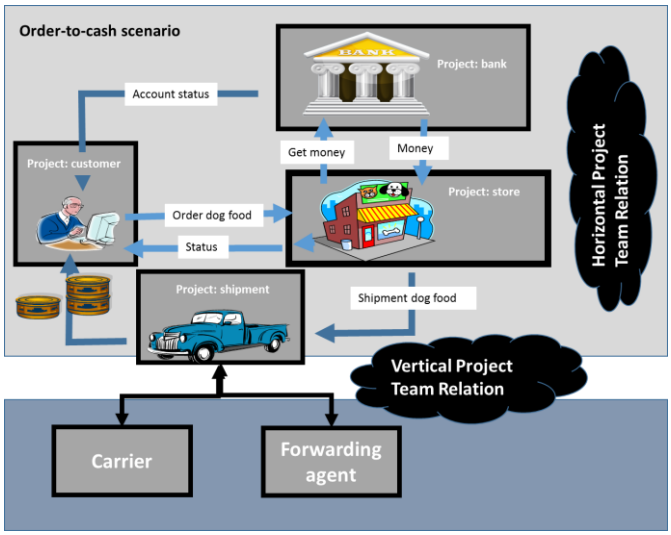

Figure 6: Multiple-team project and its communication structure

Beside that top-level communication implied by the problem structure, each team can use services offered by other enterprises. Figure 6 reveals that the shipment company uses the service of carriers and forwarding agents, in order to implement the transportation service offered to the dog food shop. This communication relation is of no interest for other federation members and thus should not be visible to the top level teams. It belongs to the internal issues of the shipment project team.

C. Development process for federated systems

In section III.A. the method for defining the functional requirements of a federated system was outlined. The artifacts to be created according to the method need to be developed by the federation of teams.

1) Specification of the communication structure

The communication between the various members of the federation needs to be specified in more detail. This is done by assigning a subject to each member of the federation and defining the messages exchanged between the subjects. Consequently, a first version of the communication structure as described in section III.A.3 emerges. Together with the data transported by the messages a communication model of the system federation is defined. The advantage of the subject-oriented approach is that the system communication structure is directly in line with the communication structure of the corresponding developing teams. The result of that step is the subject interaction diagram (SID) as shown in Figure 3.

2) Specification of the subject behaviour

After defining the communication structure the behavior of each subject is specified. The modelers describe the allowed sequence of messages exchanged on top level and the internal functions of the individual systems. These internal functions represent the services executed by the corresponding federation partner either directly or supported by other service providers. They also encapsulate the communication with those sub-contractors as it is of no interest on the top level of the federation.

The behavior of a subject is mainly defined by the corresponding project team, however, in close coordination with the teams responsible for the partner subjects. The teams only need to make sure a message sent to a partner has a receive state in the corresponding subject behavior and vice versa. This pairwise coupling means, e.g., that the behavior description of the shipment company has to contain a state for receiving the "Transfer order" message, transmitted by the related send state in the behavior diagram of the dog food store subject (see figure 5). In order to correctly model these interactions the responsible project teams need also to agree on the interaction sequence of the subjects. However, their internal task behavior (i.e. sequence of functions for task accomplishment) might not become visible to others, as is specified decentralized and might not be shared at all.

3) Implementation of the input pool

The input pool is the abstract concept for defining the semantics of message exchange. Partners exchanging messages need to agree on how they implement the input pool semantics. Sending requires the sending subject to execute a function to deposit a message in the input pool of the receiver. For each subject doing so an implementation agreement is necessary. Since an input pool is owned by exactly one subject, the functionality for accessing it is local and does not need to be coordinated with the partners. In most cases input pools are implemented as web services.

4) Implementation of subject behaviour

Each team has to implement the behavior of its subject. This means they have to ensure that depositing and removing messages (including business objects) in or from the input pool are executed and internal functions are invoked in the specified sequence. Workflow engines are appropriate tools for implementing that functionality.

5) Implementation of internal functions

The internal functions realize the kernel of the service contributed by a partner to a federated application. Messages are the means to cause the invocation of an internal function, and they transport its result to a partner subject. Internal functions can be based on existing systems, e.g., an SAP client. They also can be implemented using another federated solution, or being developed from scratch. The way an internal function is realized is a local decision taken by the corresponding project team.

6) Operation of a federated system

Beside the development and deployment the non-functional aspects of a federated system need to be agreed upon by the contributing partners. For this purpose they negotiate service level agreements (SLA) defining response time, down time, reaction time in error cases etc. The SLA also includes business aspects like costs and regulations for exceptional situations like a member leaving the federation and bringing in another one.

D. Federated work break down structure

The various activities described so far can be organized in a federated work break-down structure as shown in figure 7.

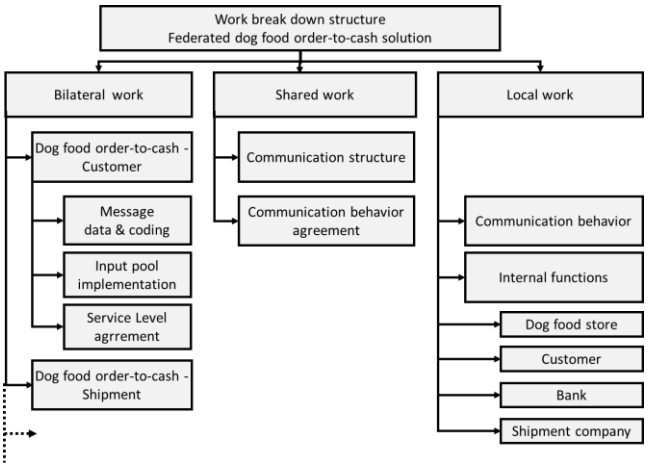

Figure 7: Work break down structure for the development of a federated system

The tasks can be divided into three types:

Joint work concerns the top level of the federation and therefore is done collaboratively by all members of a federation. The major issue on this level is to agree on communication structure and behavior of the entire system, while the behavior of each subject can be described individually by the corresponding member of the federation.

Some work can be done bilateral. Communicating partners, e.g., agree on the coding of the business objects and the implementation of the input pool. They also define the service level agreements.

Local work comprises activities of the development teams which do need to be coordinated with teams of other federation members. A major example in this context is the set of internal functions of each subject, being a local matter, and developed following the particular culture and methodology of the respective team.

E. Continuous alignment by communication

Although development can be split in joint, bilateral and local tasks accomplishment continuous communication is essential for the sustainable success of the resulting federated system.

The overall project leader and the team managers need to swiftly exchange all relevant information in order to maintain the solution according to the changing requirements of the partners.

F. Validation

The described development methodology has been partially applied in several cross-organizational industry projects. The organizations belonged to the same enterprise, which allowed a central project management. Due to restrictive permission regulation of the industry partners the publication of experiences is not possible so far. Hence, we have set up a field study to deepen our practical experiences with the presented approach.

The case deals with establishing novel services for co-housing activities, namely to support groups of people to design and implement a common housing project (see also www.tsibutsang.at, www.artsliving.at). For each housing project a project leader needs to be established, stemming either from the co-housing support provider or another project partner, e.g., a construction company. He/She ensures communication and coordination of parallel and distributed activities. For each project, various company procedures have to be aligned and a federated system has to be established.

1) Specification of the communication structure

In a workshop, the communication between the various members of the federation is specified. This step is supported by holomapping (www.vernaallee.com), as it allows identifying functional roles in a straightforward way. Once having determined them as set of holomap nodes, a subject can be assigned to each member of the federation – in the case of co-housing the cohousing service provider, the co-housing group (customer), architect (for planning), and an engineering company (for building). Moreover, the tangible relationships of holomaps represent deliverables and as such, messages exchanged between the subjects. These relationships also indicate the data transported by the messages, thus facilitating the specification of business objects, such as contracts that need to be set between the various co-housing parties.

2) Specification of the subject behaviour

Each contributing co-housing project partner (subject) has a certain behavior that needs to be detailed not only in terms of exchanging messages (i.e. the communication structure) but in terms of concrete activities (internal functions). For instance, the co-housing provider needs to arrange rooms for meetings, schedule social activities and prepare for documenting results of negotiations. These activities are mainly executed by functions provided by the cohousing providers' platform http://tsibutsang.mixxt.org/. In this case, mixxt is that federation partner's service provider. The project leader needs to ensure the completion of behavior specifications, in particular when adaptations of standard procedures, such as contracting for taking over land before authority clearance, are required,. Upon completion, the co-housing project is operationalized in terms of complete send-receive interactions between all project parties.

3) Implementation of the input pool

In that step project-specific semantics of message exchange is defined. For instance, , a message in the input pool of the co-housing support provider is based on the implementation agreement that each partner (engineering company, co-housing group) can send request messages any time with a fixed max. response time of 2 workdays. Such an agreement is due to the eventuality of social conflicts that should be addressed promptly.

4) Implementation of subject behaviour

Since in our field study workflow management systems are not being used, each involved organization needs to ensure the negotiated communication pattern with its partners. The co-housing support provider is using its mixxt-platform to trigger the (internal) project coordinator of the respective project. Hence, for each project, a dedicated workspace is established including a corresponding input pool thus enabling a dedicated communication pattern and set of business objects for each project.

5) Implementation of internal functions

Typical internal functions in the field study are requests triggering further communication or processing data by the addressed subject (project partner). For instance, a meeting of a co-housing group needs to be established, once all biddings for a certain completion step have arrived from the engineering company. Meetings are arranged invoking www.doodle.com from the meeting space of the mixxt-platform.

6) Operation of a federated system

Typical non-functional aspects of a federated system in co-housing concern the agreement to set up a task force in case of unforeseen events, selecting relevant partners to resolve all issues related to these events. Other agreements concern costs and results from regulation checks influencing original co-housing plans. Finally, changing the federation's structure in terms of membership and responsibilities is also regulated by dedicated agreements.

IV. CONCLUSION AND FURTHER WORK

We have presented an approach for developing federated systems. The concept considers the characteristics of virtual enterprises combining the services of the partners to satisfy customer needs while keeping legal, organizational, technological and cultural independence.

Our communication-oriented view follows the idea that the decentralized structure of federated systems needs to be reflected in the organizational structure of multiple project teams for developing such systems. Those teams belong to separate enterprises and are mutually independent with respect to methodology, technology etc. they use to develop their individual part of the federated system.

The proposed approach establishes a layer above the enterprise-specific environments. It helps building coherence on the top level of the federated system solution, while the teams, system elements etc. on the individual level of each federation member keep the highest degree of independence.

REFERENCES

[1] J. Putman and S. Strong, "A Federated Virtual Enterprise (VE) of Partners, Creating a Federated VE of Systems," *IEEE Proceedings of Compsac*, 1998.

[2] V. Gruhn, „Process-centered software engineering environments, a brief history and future challenges," *Anals of Software Engineering*, pp. 363-382, 14(1-4) 2002.

[3] H. F. P. Smith, Business process management—The third Wave, Meghan-Kiffer Press, 2003.

[4] L. Chengfei, L. Qing and Z. Xiahui, "Challenges and opportunities in collaborative business process management: Overview of recent advances and introduction to the special issue," *Information Systems Frontiers*, pp. 201-209, 11 2009.

[5] D. E. Avison and G. Fitzgerald, Information Systems Development: Methodologies, Techniques and Tools, New York, NY.: 3rd ed. McGraw-Hill, 2003.

[6] J. Ludewig und H. Lichter, Software Engineering, 3. Edition, Heidelberg: d-punkt Verlag, 2013.

[7] V. Stiehl, Prozessgesteuerte Anwendungen entwickeln und ausführen mit BPMN, Heidelberg: d-punkt Verlag, 2013.

[8] R. Cognini and et. al., "HawkEye: A tool for collaborative Business Process modelling and verification," in *SAC'13*, Coimbra, Portugal, 2013.

[9] L. S. Sterling and K. Taveter, The Art of Agent-Oriented Modeling, Cambridge, Massachusetts: The MIT Press, 2009.

[10] R. Ramsin and R. F. Paige, "Process-centered Review of Object Oriented Software Development Methodologies," *ACM Computing Surveys*, Vols. Volume-40, Number 1, 2008.

[11] M. E. Conway, "How do Committees Invent?," vol. 14, 1968.

[12] J. O. Coplien and N. B. Harrison, Organizational Patterns of Agile Software Development, Prentice Hall International, 2004.

[13] T. Walke and et. al., "Case Study @ Swisscom (Schweiz) AG: iPhone 5 Self-Service Order App and Process-Workflows," in *H. Fischer, J. Schneeberger (Ed.), S-BPM ONE - 2013, CCIS Vol. 360*, Heidelberg, Springer Verlag, 2013.

[14] G. Konjack, „Case Study: AST Order Control Processing," in *H. Buchwald et.al. (Eds.), S-BPM ONE - Setting the stage for Subject Oriented Process Management, CICS Vol. 85*, Heidelberg, SPringer Verlag, 2010.

[15] S. Nakamura and e. al., "CGAA/EES at NEC Corporation, Powered by S-BPM: The Subject-Oriented BPM Development Technique Using Top-Down Approach," in *W. Schmidt (Ed.), S-BPM ONE - Learning by Doing - Doing by Learning, CCIS Vol. 213,*, Heidelberg, Springer Verlag, 2011.

[16] A. Fleischmann, U. Kannengiesser, W. Schmidt and C. Stary, "Subject-Oriented Modeling and Execution of Multi-Agent Business Processes," Atlanta, 2013.

[17] A. Fleischmann, W.Schmidt, C. Stary, S. Obermeier and E.Börger, Subject Oiented Business Process Management, Berlin, Heidelberg: Springer, 2012.

[18] R. K. Wysocki, Effective Project Management, Seventh Edition, Indianapolis: John Wileys & Sons, 2014.

[19] E. Yu und et. al., Social Modeling for Requirements Engineering, Cambridge, Massachusetts: The MIT Press, 2011.