Light-weight Model-based Realization of a B2B Protocol and a SOA Integration Engine

Theo Dirk Meijler
SAP Research Dresden
theo.dirk.meijler@sap.com

Frank Nietzold
SAP Research Dresden
frank.nietzold@sap.com

ABSTRACT

In emergency management, communication between the emergency management team and the outer world is essential. When using an emergency management system, such communication is often IT-based. To disburden the emergency management team, structured "B2B" messages may be used that correspond to the (foreseen) lifecycle of relevant entities in the emergency, such as threats and measures. The paper introduces an approach for the realization of a B2B messaging protocol and the corresponding integration engine, which maps message content to service calls, in the context of an emergency management system. The approach is light-weight and model-based, as protocols and integration engine are based on merely modeling the states and state transitions of objects in the system representing essential entities in the emergency. As the model is described in non-technical terms, this can be done by a non-IT expert.

Keywords

Interoperability, messages, communication, service-oriented architectures, ontologies, B2B.

INTRODUCTION

An organization responsible for managing an emergency –the so-called emergency management team (EMT)– may use a so-called "emergency management system" (EMS) (Braune, Brucker, Kleser, Keqin, Meijler, Paulheim and Probst, 2011) to get an overview of the situation, bring together all information about the emergency, provide decision support on appropriate measures, support planning of such measures, track their execution etc. Of course this requires communicating with agencies handling the emergency at location and the general public. To optimally use an EMS (e.g. for tracking history), IT-based communication must be used as much as possible, or communication must be mapped to IT-based communication. Such communication may be relatively *unstructured* (freeform) (Underwood, 2010). However, given the sheer mass of communications entering and leaving an EMT, team members must be disburdened from handling all these communications. *Structured message based communication* can play an important role in this respect. Structured messages can describe specific pre-defined occurrences (such as "resource arrived" or "measure completed") in the emergency in a fixed form as linked with pre-known entities represented in the EMS, such as threats, damages, measures (Sell and Braun, 2009) and resources. Since structured message based communication may be automatically associated with entities represented in the EMS, such automated association may simplify (and sometimes even remove) the need for human intervention when processing such messages, for example, when a message indicates the normal progress of the execution of a certain measure.

Structured message based communication is known from the application domain independent technology of "organizational" communication and negotiation (Medjahed, Benatallah, Bouguettaya, Ngu and Elmagarmid, 2003). Given the literature, we speak of Business to Business (B2B) message based communication (B2B-MBC). Typically, in B2B-MBC complex protocols can be followed (Bussler, 2001), for example concerning contract negotiation (Schoop M. et.al., 2003). Setting up a B2B-MBC is currently often a lengthy process, requiring agreement on the protocol, and a subsequent implementation of a B2B *Integration Engine* (Bussler, 2001) that takes care of processing the messages and storing the intended impact in the local system. Often, this is based on a standardization effort. Specialization of the standard to individual needs of collaborating partners is therefore not or insufficiently supported (Van Blommestein, 2005; Bussler, 2001). In fact (Bussler, 2001), strongly urges the need for B2B Protocol and Integration Engines that are agnostic to the B2B and where new

Reviewing Statement: This full paper has been fully double-blind peer reviewed for clarity, relevance, significance, validity and originality.

B2B protocols can dynamically be added.

For the use within EMSs, the fact that realizing a B2B protocol and a corresponding integration engine requires considerable effort is problematic as this burdens the development of these systems, and the introduction of new collaborative scenarios in such systems. The contribution of this paper is therefore, that it describes a light-weight approach –i.e. light-weight in terms of effort as compared with other existing approaches– towards realizing B2B message based protocols of structured messages. The *purpose* is therefore to minimize the effort to set up the structured communication protocol about any emergency relevant entity as represented in an (any) EMS, be it a measure, a resource, a flooding threat etc. The *underlying idea* is that we view such protocols as communications about the state and state transitions of the represented entity and by declaratively modeling the state transition graphs (STG) of such entity representations and linking this to corresponding messages.

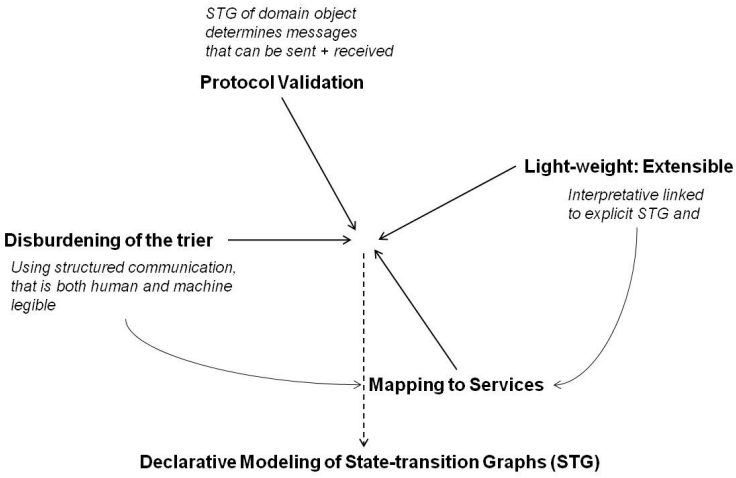

Figure 1 Main features at a Glance

Our approach is based on an interpretation of declarative STG model information, through this extensibility and even run-time extensibility is achieved, although the run-time aspect is not the main focus. As summarized in Figure 1 our contribution features furthermore disburdening the trier, the person in the EMT who views messages and dispatches them to the members that have to handle the messages. This is done through communication that is both human and machine legible. The former is clearly required in emergency management. The latter allows not only for automatic association of the message to an entity representation as mentioned above, it allows for automatic run-time protocol validation (based on the same declarative STG model information) and mapping message contents to service calls of the EMS. The approach includes the largely interpretative realization of a B2B integration engine. Declarative model information is used to describe how the messages can be translated into service calls. Our approach has been realized and applied in the SoKNOS Emergency Management System (Braune et al., 2011). In the SoKNOS EMS, messages will be directly sent and received by a squad leader who is at location using his / her smart mobile phone.

The document is structured as follows: Section 2 describes background –knowledge which is necessary for understanding the approach– and related work. This includes a further explanation about the need for using structured messages in emergency management, and the use of the Language Action Perspective (LAP) as the main foundation for our message protocols. In section 3 we present a typical B2B scenario as used in an emergency management system including the messages which have to be exchanged. The approach itself is described in section 4, first as an overview and afterwards in detail. Section 5 concludes this paper.

BACKGROUND AND RELATED WORK

Emergency Management and the need for structured messages

Emergency Management Systems (sometimes also called "Disaster Management Systems"), such as IGNIS (Grafling, 2001), Sahana (Treadgold, 2006) and the SoKNOS Emergency System (called S3 for "SoKNOS Support System") (Braune et al., 2011) in which context this work was developed, need to support communication between the Emergency Management Team (EMT) and the outer world.

The most work done so far on explicitly supporting and processing communication in EMSs has been in the area of unstructured communication, thus the handling and processing of free text information transmitted via

common communication mediums, such as from twitter, sms etc. (Underwood, 2010) or of images from the emergency situation (ImageCat, 2011; Underwood, 2010).

The use and need for structured (message-based) communication in EMSs has the following background: A relevant interaction form between *any* (IT or non-IT supported) EMT and the external world is in the form of (textual) messages. In fact, the functioning of an EMT that is not supported by IT is for a larger part based on procedures how to handle such messages (Endres, Wurz, Hoffmann and Behring, 2010.). The trier views messages and dispatches them to the members that have to handle the messages. Messages can contain any kind of information about the emergency. Given the huge number of incoming and outgoing messages, any support to disburden the trier and other EMT personnel is of relevance.

The S³ (but for that matter, any EMS) maintains information about relevant entities in the emergency, such as the threats, damages, available resources, measures (activities) etc. Within the S^3 information that is maintained about such entities is represented as data objects, in SoKNOS we call these "domain objects". For many domain objects a specific (possibly complicated) lifecycle may be defined of possible statuses and status transitions of the relevant entity in the emergency that the domain object represents. For example a resource such as a fire brigade truck can be moving, arrived, broken, a measure (activity) can be started, running, delayed, completed etc. Structured messages then encode such situations. Using the structured messages it then becomes possible to directly (without intervention of the trier) associate a message with a certain representing domain object, dispatch the message to the responsible coordinating officer within the EMT, and possibly, e.g. in case of "normal" progress message, the need for human intervention can be removed. (Sell and Braun, 2009) have indeed shown that it is possible to use such structured messages for tracking the status and transferring commands concerning certain measures (in fact called "activities"). Sell and Braun have, however, not described how to realize the corresponding protocol and integration engine.

The use of task-based categorization of Endres et.al. (Endres et.al., 2010) is another mechanism for disburdening the trier, their work is not based on structured messages, but on automatic (free-text) message analysis, and corresponding derivation of the requested task, and therefore simplifying dispatching the message to the responsible EMT member who has to handle that task.

B2B Protocol Realization

As far as we know, in the current work on EMS and EMS realization, no specific work has been focused on realizing protocols for structured messaging. For this, we refer to work on application-domain independent B2B message-based communication (B2B-MBC).

One important tool for coding structured messages describing what is happening in the real world is the so called Language Action Perspective (LAP) (Schoop, 2001). LAP is an approach based on linguistic theories. It was first introduced by Flores & Ludlow in 1980 (Flores and Ludlow, 1980). Two theories from linguistics form the foundation of the LAP: The Theory of Speech Acts (Searle, 1969) and The Theory of Communicative Action (Habermas and McCarthy, 1984). As our approach makes use of the Theory of Speech Acts to model message types we explain this theory here in detail.

Searle argued that language consists of *speech acts* as basic units of an utterance. With each utterance the speaker performs an action (speech act). For example with the expression "I beg your pardon?" the speaker requests his conversational partner to repeat the last sentence or by expressing the wedding vow a wedding couple changes their marital status. Each speech act consists of two components, which together form the meaning of an utterance: the *propositional content* and the *illocutionary force*. The propositional content describes the actual content of an utterance, i.e. what the utterance is about, e.g. move tactical unit to location x until next Tuesday. The illocutionary force describes the way it is uttered or the speakers intention, e.g. the propositional content above can be formulated as statement ("Someone is moving the tactical unit to location x until next Tuesday."), as request ("Could you please move the tactical unit to location x until next Tuesday?") or as promise ("We will move the tactical unit to location x until next Tuesday."). Searle classifies speech acts into five categories:

- *Assertives*: inform about facts of the real world (e.g. reports)
- *Directives*: the speaker wants the listener to perform an action (e.g. requests)
- *Commissives*: the speaker itself declares to perform an action (e.g. promises)
- *Expressives*: express the speaker's internal feelings or attitudes (e.g. apologies)
- *Declaratives*: change the state of the world by the utterance (e.g. a wedding vow)

The illocutionary force of one category can have different degrees of strength. For example, a directive illocutionary force can be weak (e.g. a request) or strong (e.g. an order).

DOC.COM (Schoop. and Quix, 2001) applies the LAP to support of complex electronic negotiations among human negotiators, which was designed especially for the domain of B2B e-commerce. *Negoisst* (Schoop, Jertila and List2003) is the corresponding implementation of DOC.COM. In the approach a negotiation protocol is defined based on the illocutionary message types *request*, *offer*, *counter-offer*, *accept*, *reject*, *question* and *clarification*. However, Schoop et.al., do not describe how in general to define protocol on basis of LAP.

Thus, while LAP provides a relevant approach how to code messages, it does not provide a method for defining and realizing a specific protocol and the corresponding integration engine. Bezivin et.al. (Bezivin, Hammoudi, Lopes and Jouault, 2004) have described a Model-driven approach to realizing B2B protocols. Although a Model-driven approach is principally meant to simplify the realization of a system, the authors themselves indicate the complexity of the models they use.

We assert that the problem of model complexity, that Bezivin et.al. have indicated is a result of the fact that they apply the MDA (Model-driven Architecture) which is again based on a general purpose modeling language such as UML. In concert with (Cook, 2004) we believe that a domain-specific modeling language is needed, a modeling langue that is "carefully designed to facilitate modeling within a particular problem domain". Thus, "light-weight" modeling as we propose this, can be achieved when a modeler needs only to model what is relevant to vary, and modeling is not needed for those aspects of the solution that are fixed. In fact, the fixed part will be encoded by the underlying *software framework* (Cook, 2004): "Such a framework is a body of code that implements the aspects that are common across an entire domain".

MAIN EXAMPLE

As mentioned before, this work has been developed and realized in the context of the SoKNOS project. Our main example has also been taken from this context.

With respect to measures (activities) the SoKNOS system encompasses a "Process Engine service" (Sell and Braun, 2009). This service not only maintains information about which measures were or are to be taken during the emergency. It maintains information about the planned sequence of these measures and their progress. The Process Engine service is used as a kind of workflow engine to support the emergency team to carry out a complete plan, the process engine service can handle updates on the state, and thus monitor the progress of the plan, indicate which next measures must be executed etc. Since the Process Engine is really a workflow engine, we also speak about "activities" instead of measures, as these activities are executed and monitored in order to handle an emergency.

In our main example messages are exchanged about the status of certain activities between a coordinator (part of the EMT), who is fundamentally empowered to plan certain activities and control their execution using the EMS, and an executing party (executor), who basically follows the directives of the coordinator and provides the coordinator with information, both on the situation at location as well as on the progress of the activity.

The EMS maintains the activity domain object representing an activity at the scene of the emergency as carried out by the executor. When the coordinator wants a local team to execute a certain activity, for instance the building of a sandbag wall, the staff can plan and start this activity with the help of their EMS. The activity domain object initially gets the status "activated". At this point the executor must be informed about the new activity and its execution must be requested. For this reason a message containing all the required information has to be sent to the executor. The EMS generates a message and sends it to the executor. The executor has to initialize the execution of the activity at the location of the emergency. Having started the execution the executor sends a confirmation message back to the coordinator. The EMS of the coordinator updates the state of the domain object accordingly, expressing that the execution is now actually in the status "running".

While the activity is running, it can happen that the coordinator wants information about the progress of the activity. Through the EMS a "request for progress" message can be generated. The EMS sends that information request to the executor, who has to reply with the current progress information. Using the answer message the EMS can update the activity's domain object. This part of the protocol can be repeated several times.

When the activity is finished in the external world, the executor informs the coordinator accordingly. As a result the EMS sets the status of the activity's domain object to "completed".

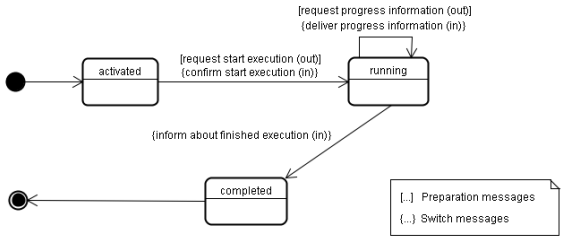

Figure 2: A message protocol for the execution of an activity

The described scenario is shown in Figure 2 through a state-transition-graph (STG) of the domain object. Each state transition contains messages labeled with "out" that are sent from coordinator to the executor, and messages labeled with "in" that are sent, possibly as a reply, from executor to coordinator.

APPROACH

Overview

Our approach is based on the idea that messages are exchanged between coordinator and executor in order to communicate about state changes of domain objects (see Figure 3). This information is transferred from the real world to the EMS and vice versa. We formalize the message exchange by defining a fixed message structure to encode all information required for transferring a state change of a domain object. Each structured message has a type. Possible message types are based on the speech act theory of the LAP in order to encode the communication about the state change, e.g. requesting, confirming or denying a state change. Possible protocols of messages are described by modeling the STG of domain objects, that is in terms of states and stage changes.

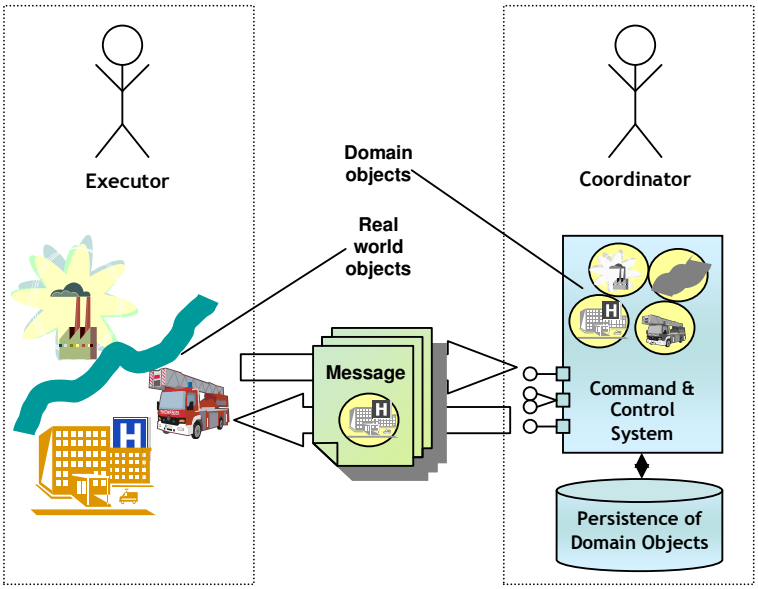

Figure 3: Message exchange between coordinator and executor

The EMS manages the domain objects and offers services for changing them. Therefore the messages have to be mapped to service calls of the EMS. For this purpose we also present a mechanism for describing these mappings.

Both message protocols through STGs as well as message-to-service mappings are modeled with the help of ontologies enabling domain experts to extend the solution by new domain objects and correspondingly new or adapted message protocols.

We have designed and implemented a framework using those ontology based models for processing incoming and outgoing messages. Figure 4 shows an overview over the framework architecture. The Message Information Handler is the central component of the framework. It accepts messages for processing and calls the other

framework components. The processing itself consists of three basic steps (the framework is explained in more detail in section 4.2.4.):

1. Execute the mapping to find out, which service is to be called (Done by the Semantic Processor using a reasoner).
2. Check whether the message is valid for the current state in the protocol (Validator, State Manager).
3. If validation is positive, call the service (Service Caller).

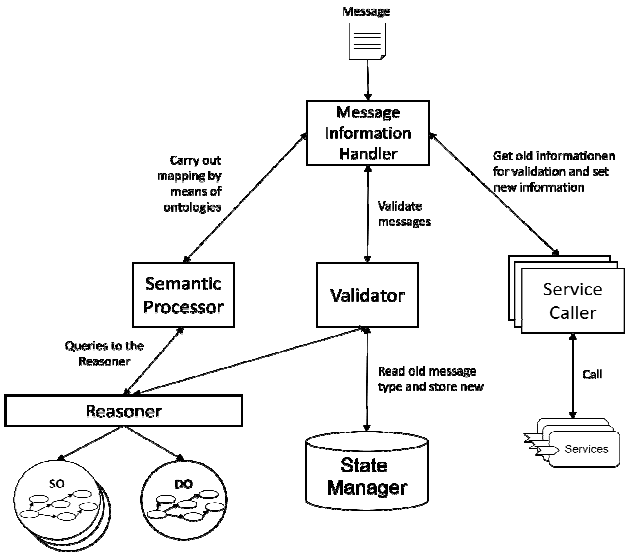

Figure 4: Overview over architecture

Details

Describing Messages

According to speech act theory each utterance consists of an illocutionary force and a propositional content. As a message is a form of an utterance, we apply this theory for formalizing our messages. Based on the illocutionary force of the speech act theory we determined a fixed set of message types expressing the intention behind the message concerning the state change of a domain object. These are the following (the speech act category is given in brackets):

- **Request state change**: ask for a state change with the receiver having the option to confirm or deny the request (*directive*)
- **Order state change**: corresponds to a command that must be executed by the receiver, there is no choice (*directive*)
- **Confirm state change**: expresses the intention to perform an action, can follow to a request or an order (*commissive*)
- **Inform about state change**: informs the receiver about a new state of a domain object (*assertive*)
- **Deny state change**: expresses the intention not to perform a requested action (*commissive*)
- **Inform about new object**: informs the receiver about a new object in the external world (*assertive*)
- **Request information**: asks for information about a domain object (*directive*)
- **Deliver information**: delivers requested information about a domain object (*assertive*)

These message types are specified in a domain ontology and can be used for describing message protocols (section 4.2.2) and message-to-service mappings (section 4.2.3).

According to speech act theory each utterance moreover consists of a propositional content describing the actual content of the utterance. Applied to messages concerning state changes of domain objects we determined the following message structure:

- *message type:* one of the types specified above, e.g. “request state change”

- *object type:* type of the domain object, e.g. "activity"
- *object reference:* concrete domain object, e.g. "building sandbag wall at location …"
- *property:* single property of a domain object, e.g. "status"
- *old value:* old value of the property that should be changed, e.g. "suspended"
- *new value:* desired new value, e.g. "failed"
- *time slot:* time of the state change, e.g. "immediate"
- *reason:* a free text, e.g. "no longer important"

Describing Message Protocols

As mentioned earlier, we provide a mechanism to define message protocols, which describe permitted message sequences. A message protocol is defined on bases of the set of states and a set of state transitions of a domain object. Each state transition is assigned at least one message type, which triggers the state change. Figure 2 in section 3 shows the protocol for the execution of an activity as described.

We distinguish between preparation messages and switch messages. Only switch messages trigger a state change, preparation messages do not trigger any state changes but can be preconditions for a switch message. For example a message of the type *confirm state change* switches the activity state from *activated* to *running*, but this message will only be accepted, if a *request state change* message had been processed before.

Such message protocols are modeled in a domain ontology which can be relatively easily be extended by a (non-IT expert) modeler, allowing new protocols to be added or existing ones to be edited. Figure 5 shows a graphical representation of the ontology representation of a part of the message protocol presented in Figure 2.

. Each state of the domain object, being also a state in the protocol, is explicitly specified as a sub concept of the domain object property, which is to be changed by a message. That is *StatusActivity* in the example. This property has to be in relation *involvesDomainObject* to the concept describing the type of domain object, which is *Activity* in the example. Furthermore every state transition is explicitly modeled as a sub concept of a concept representing a domain object specific state transition (*StateTransitionActivity*), which is again a sub concept of the general *StateTransition* concept. In Figure 5 the state transition *Activated-Running* is used as an example for all state transitions of the protocol.

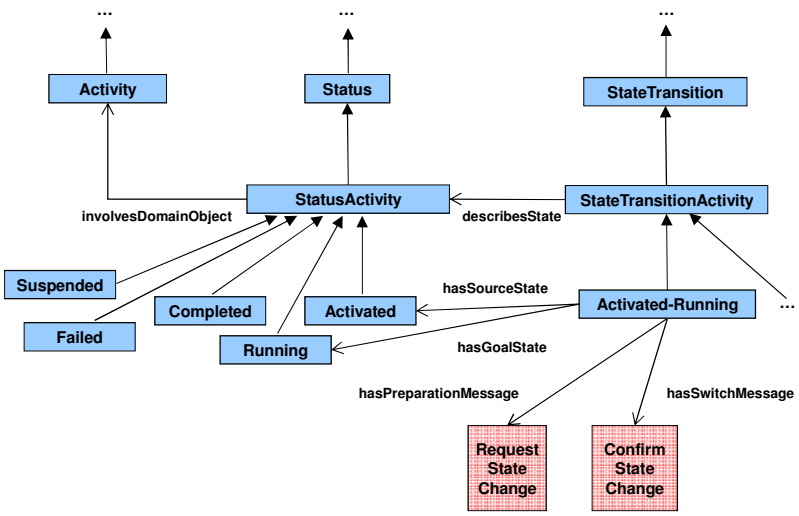

Figure 5: Modeling a message protocol in an ontology

Describing Message-to-Service Mappings

Besides providing a mechanism for modeling message protocols via ontologies our approach supports a mechanism for describing via ontologies how messages are mapped to service calls. Here, too, the use of ontologies makes it relatively simple for a modeler to extend the model and add new mapping rules.

Principally, we model for each operation of a service what kind of message (including its contents) can trigger the call to that operation. In more detail, we model the connection of each operation of each service to the domain object type, the type of the domain object property and the message types, which can trigger the service

operation. Figure 6 shows the mapping rule for messages concerning the status of activities as described in the main example. Each service has to be modeled as a sub concept of the concept *Service* and each service operation has to be a sub concept of one of the concepts *Get*, *Update* or *Create*, denoting operations for reading a domain object property, updating it or creating a domain object. In the example there is the service *ProcessEngine* providing operations for updating and getting the status of an activity. The *UpdateActivityStatusOperation* can either be triggered by a message of the type *Request*, *Confirm* or *Inform About State Change*, the corresponding get operation is automatically triggered together with the update operation, as it has to be called prior to it for reading the old status value, which is needed for validation. We decided to model this in a new independent service ontology, as this is rather technical information in contrast to the domain information which is modeled in the domain ontology, e.g. the message types and the message protocols. There can be several service ontologies, usually one for each service.

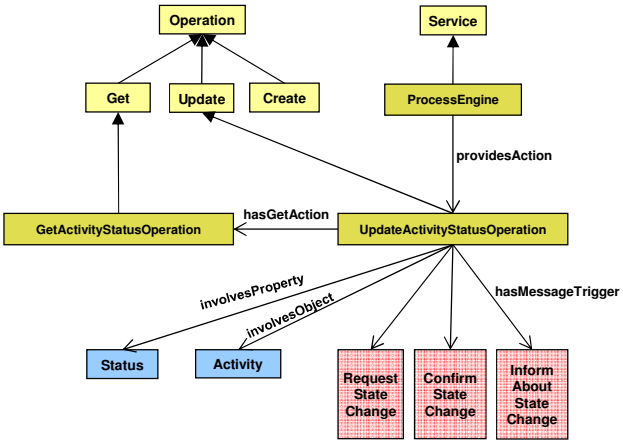

Figure 6: Modeling the message-to-service mapping in an ontology

The Components of the Framework in Detail

So far we have explained how to model all the information that is required for a message-to-service mapping and for the validation against a message protocol. In this section the framework is described as a whole putting all those pieces together for fulfilling its task, namely the automatic message handling for updating the state of a domain object.

Figure 4 shows all components of the framework. As already described the Message Information Handler is the central component, which is responsible for accepting messages and calling the other components. The Service Caller encapsulates all the code that is required to execute a service call. Since there may be different ways for accessing different services even within one EMS, there is a specific Service Caller for each service that decouples the Message Information Handler from this heterogeneity. Service Callers provide a standardized interface for invoking heterogeneous service operations. This interface has the domain object reference and the property value as parameters.

The Semantic Processor is responsible for finding the appropriate Service Caller, after getting the message as input. The mapping to a service call works as follows: When a message is handled, the parameters *message type*, *object type* and *property* are read from the message. On basis of that information the Semantic Processor creates a query to the reasoner to search through the service ontologies for an operation that involves those object and property types and is triggered by that message type. If such an operation is found the Semantic Processor returns the names of the service and the operation to the Message Information Handler. These names are equal to the names of the respective Service Callers and their operations, so the Message Information Handler can create a call to the respective Service Caller operation. This is implemented using reflection. The parameter values of the Service Caller operation can be directly copied from the *object reference* and *new value* parameters of the message.

Before the actual service call can be executed, the message must be validated against the message protocol. This is the validator's task. It again uses the reasoner in order to obtain the message protocol which belongs to the domain object and property type. With the information of the current state of the domain object, which is gathered by a prior service call, and the type of the lastly handled message, which is stored by the State Manager

component, the actual position in the protocol is determined and the validator can check, whether the current message type and the new state of the domain object are permitted at this point.

The message parameters *time slot* and *reason* are not needed for the automatic handling, these are merely used as additional information for human readers.

CONCLUSION

This paper introduced an approach for the realization of a B2B protocol and the corresponding integration engine, as realized and applied in the SoKNOS project.

The advantages of the presented approach are as follows:

- By applying structured messages, EMT personnel can be disburdened, as structured messages can be automatically associated with specific domain objects in the EMS. Thus the dispatching of messages to responsible personnel can be automated, moreover, in case of messages representing "normal progress", human intervention can even be removed
- The realization of the protocol and the corresponding is "light-weight" as the modeling is limited to the aspects that are variable for realizing such a protocol, namely the states and state transitions of the domain object, how state transitions are linked to messages, and which service calls –implementing the integration engine– must be called for processing which messages. This in contrast to general purpose modeling approaches such as described by (Bezivin et.al., 2004) and in agreement with the statements of (Cook, 2004).
- The approach can be applied for describing protocols around any kind of domain object representing relevant entities in an emergency, and since any EMS presumably stores and represents such information about relevant entities, it should be applicable in any EMS.
- Due to the light-weight modeling, the provided approach offers a mechanism for introducing new collaboration scenarios around domain objects, thus supporting a good "design-time" for implementing structured messaging in an EMS.

For future work we intend to extend our work for describing state transitions and corresponding messages of domain objects that not only express the collaboration between two organizations (EMT and executor in this case), but between multiple organizations.

REFERENCES

1. Bezivin, J., Hammoudi, S., Lopes, D., Jouault, F. (2004). Applying MDA approach to B2B applications: A road map. In: *Proc. of theWorkshop on Model Driven Development at the 18th European Conf. on Object-Oriented Programming*.
2. van Blommestein, F. (2005). Decentralized metadata development for open B2B electronic business. *Proceedings of the 2005 international conference on Dublin Core and metadata applications: vocabularies in practice* (pp. 1--7). Dublin Core Metadata Initiative.
3. Braune, S., Brucker, A.D., Kleser, G., Keqin, L., Meijler, T.D., Paulheim, H., Probst, F. (2011). A Service-oriented Architecture for Emergency Management Systems. *Innovative Systeme zur Unterstützung der zivilen Sicherheit: Architekturen und Gestaltungskonzepte,* GI
4. Bussler, C. (2001). B2B protocol standards and their role in semantic B2B integration engines. *Bulletin of the Technical Committee on Data Engineering*, Vol. 24 No. 1.
5. Cook, S. (2004). Domain-Specific Modeling and Model Driven Architecture. *MDA Journal,* 1–10
6. Endres, C., Wurz, A., Hoffmann, M., Behring, A. (2010). A Task-based Messaging Approach To Facilitate Staff Work. *Proceedings of the 7th International ISCRAM Conference--Seattle*
7. Flores, F. and Ludlow, J.J.. (1980). Doing and Speaking in the Office. *ISSUES AND CHALLENGES*, pp. 95-118.
8. Grafling, W. (2001). IGNIS-die neue Leitstelle der Berliner Feuerwehr. *Brandschutz-Deutsche Feuerwehrzeitung*, 55,4:449–455, Stuttgart: Verlag W. Kohlhammer GmbH
9. ImageCat (2011). http://www.imagecatinc.com (Last call: 14-03-2011).
10. Habermas, J. and McCarthy, T.. (1984). The theory of communicative action: Reason and the rationalization of society. Boston: Beacon Press.

11. Medjahed, B., Benatallah, B., Bouguettaya, A., Ngu, A.H.H. and Elmagarmid, A.K. (2003). Business-to-business interactions: issues and enabling technologies. The *VLDB Journal (2003)* 12, pp 59–85
12. Schoop, M. and Quix, C. (2001). DOC. COM: a framework for effective negotiation support in electronic marketplaces. *Computer Networks , 37* (2), pp. 153--170.
13. Schoop, M., Jertila, A. and List, T. (2003). Negoisst-A Negotiation Support System for Business-to-Business Electronic Commerce. *Data Knowledge and Engineering, in print* .
14. Schoop, M. (2001). An introduction to the language-action perspective. *ACM SIGGROUP Bulletin , 22* (2), pp. 3--8.
15. Searle, J. (1969). Speech acts: An essay in the philosophy of language. Cambridge University Press.
16. Sell C. and Braun, I. (2009). Using a Workflow Management System to Manage Emergency Plans. *Proceedings of the 6th International ISCRAM Conference.* Gothenburg, Sweden J. Landgren, U. Nulden and B. Van de Walle, eds.
17. Treadgold, G. (2006). Sahana-Engineering a sustainable ICT solution for disaster management. *Digital Earth*, 6:27–30.
18. Underwood, S. (2010). Improving disaster management. *Commun. ACM, 53,* pp. 18--20