Negotiation-based Patient Scheduling in Hospitals

Reengineering Message-based Interactions with Services

Lars Braubach and Alexander Pokahr and Winfried Lamersdorf

Abstract Nowadays, hospitals in Germany and other European countries are faced with substantial economic challenges stemming from increased costs and increased demands inter alia resulting from a changing age pyramid. In this respect, patient scheduling is an interesting parameter that determines on the one hand the length of the patients stay in the hospital and the efficiency of the hospital resource allocation on the other hand. Due to the specific characteristics of hospitals such as unexpectedly arriving emergencies or unintended complications during treatments the optimization of patient scheduling is an extraordinary difficult task that cannot be easily solved using a typical centralized optimization algorithm. Thus, in this paper a decentralized agent based approach is presented that represents patients as well as hospital resources as agents with individual goals that negotiate to find appropriate solutions. The approach has been extensively studied within the MedPAge project and also has been implemented using an agent platform. In this work it will be shown how the traditional message based implementation, which was difficult to construct and even more difficult to maintain, can be replaced with a service based design.

1 Introduction

In hospitals, patient scheduling deals with the assignment of patients to the scarce and expensive hospital resources. Optimizing patient scheduling is able to reduce hospital costs due to a reduced stay time of patients within the hospital and an increased capacity utilization of hospital resources. Despite its

Distributed Systems and Information Systems Group,
Computer Science Department, University of Hamburg,
Vogt-Kölln-Str. 30, D-22527 Hamburg, Germany,
e-mail: braubach|pokahr|lamersdorf@informatik.uni-hamburg.de

general usefulness, efficient mechanisms for patient scheduling are difficult to devise due to some inherent characteristics of hospitals. Foremost, in hospital many uncertainties exist that make precise planning ahead for weeks or even just days very difficult if not impossible. These uncertainties e.g. result from unexpectedly arriving emergencies and treatment complications and may require far-reaching changes with respect to the patient scheduling plans. Another important difference with respect to other application domains with rather static workflows consists in the unpredictability of patient treatment processes within the hospital. Although approaches like clinical pathways [5] aim at standardizing the treatment steps of patients for certain kinds of diseases, patient cure remains a task that cannot be preplanned completely in advance. Thus patient scheduling has to accommodate these characteristics and provide a flexible and fast mechanism that is able to handle uncertainties at many levels.

Within the MedPAge project [7, 14, 9], patient scheduling is treated as a decentralized coordination problem between patients and hospital resources. Both types of stakeholders are modeled as autonomous decision makers with their own selfish goals that need to come to agreements in order to build up scheduling plans. In the following, the foundations of patient scheduling in hospitals will be described in Section 2. Afterwards, in Section 3, the MedPAge approach will be introduced by explaining its coordination mechanism, the general system architecture and its implementation based on multi-agent system technology. Implementation experiences showed that a message based realization of the complex coordination protocol is difficult and error prone. Hence, in Section 4 an alternative coordination description and implementation based on service interfaces is proposed and illustrated by dint of the MedPAge coordination mechanism. Section 5 summarizes and concludes the chapter.

2 Foundations

Hospitals typically consist of different rather autonomous units that are responsible for different medical tasks. Patients normally reside at wards and visit ancillary units according to prescribed treatments. Often, these treatments are prescribed by physicians as part of the ward round taking place in the morning of each day. The treatment requests are announced to the different ancillary wards, which decide on their own behalf at what time a patient will be served. For this purpose the ancillary unit typically employs a first come first served principle and calls the patients from the ward in the order the requests arrived. This scheme is very flexible and allows the ancillary wards to react timely to unexpectedly occurring emergencies by calling an emergency before normal requests and with respect to complications by simply delaying the next patient call as long as needed. Besides its flexibility

the scheme also incurs drawbacks that derive from the local unit perspective used. This local view does not take into account global constraints e.g. the current allocations of other ancillary units or the overall treatment process of a patient.

Patient scheduling in hospitals aims at optimizing the temporal assignment of medical tasks for patients to scarce hospital resources with two objectives. On the one hand the patient's stay time and on the other hand the resource's idle times should be minimized. As both objectives are not necessarily congruent, a coordination approach can be used, in which both sides try to reach agreements by making compromises. In order to negotiate about the different time slots at scarce hospital resources the patients need to be able to quantify their interest in the resources. For this purpose the preferences of patients should be expressed in terms of the medical priority of the examinations or treatments for the patients. In this respect a concept for opportunity costs has been developed that is able to assess the difference of the patients health state with and without a potential treatment taking into account the duration of that treatment. In this way the potential decrease of a patients health state is used as a measure for the criticality of the treatment for that patient. As durations of treatments can vary in practice stochastic treatment durations have been taken into account. Details about the design of medical utility functions can be found in [7].

3 MedPAge Approach

A major objective of the MedPAge project consisted in finding an adequate decentralized coordination approach that on the one hand respects the decision autonomy of the units and on the other hand is able to generate anytime solutions that are at least near to pareto-optimal assignments. In order to come closer to this aim different coordination mechanisms [8, 9, 6] have been designed and benchmarked with respect to the current practice in hospitals. The benchmarking was based on a simulation system that used collected real world hospital data from a middle sized German hospital. The benchmarking revealed that an auction based scheme similar to contract-net performed best and was able to outperform the current hospital practice from 6 to 10 percent depending on the number of emergencies occurring (the more emergencies the lower the gain) [7]. Furthermore, the approach had been field tested in a hospital and was considered as a helpful decision support system by the staff [14]. In the following the details of the conceived coordination mechanism will be described (cf. [7]).

3.1 Coordination Protocol

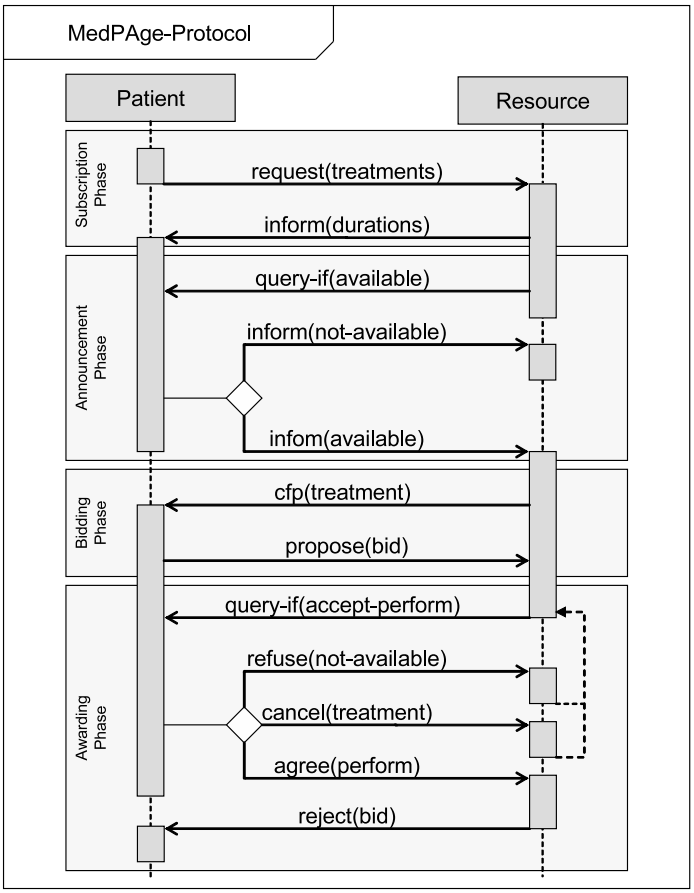

Fig. 1 MedPAge coordination mechanism

In general, each hospital resource auctions off timeslots for treatment and examinations in near real-time, i.e. shortly before the ongoing treatment has finished and the resource expects to become available again. A time slot is assigned to the patient that placed the highest bid according to the health-state based utility functions introduced in the last section. The coordination protocol is designed to have four phases (cf. Figure 1).

During subscription phase patients first have to find resources fitting to the treatments or examinations currently needed. Due to the fact that multiple medical units may offer the same treatments or examinations, a patient can subscribe at several units at the same time. Although the hospital infras-

tructure is considered as quite stable in comparison to the continuous arrival and dismissal of patients, it is assumed that from time to time changes can occur, e.g. a new medical unit is added or the operation of a unit is temporary closed. For this reason patients should always perform a fresh search for adequate resources and afterwards subscribe at the corresponding resources. A resource answers to the patient with an estimate of the duration for the tasks requested. These durations are deduced from historical data and consist of mean and variance to cope with their stochastic nature.

In the announcement phase a resource initiates a new auction whenever the ongoing medical task is about to finish. In this way the flexibility of the current practice in hospitals is kept and possibly occurring disturbances caused by emergencies or complications can be addressed in the same way as before. The resource agent announces the auction by requesting which patient is currently willing to participate. This could be a subset of the subscribed patients as some of them could already have won an auction at another resource.

After having determined the participants, a call for proposal is sent to them with detailed information about the auctioneered treatment including the expected duration. Afterwards the patients are expected to send bids according to their utility functions, i.e. patients determine their bids according to the expected utility loss if they would loose the auction. This disutility is calculated by comparing the health state at the assumed finished time with and without the treatment.

The resource collects the incoming bids and sorts them according to the bid value. It will then start determining the winner in the awarding phase by notifying the first patient from the list. If the patient is still available it will agree and the resource will inform the other patients about the auction end. If not, the resource will move on with the next patient from its list and continue in the same way as before until a winner could be determined or no more patients are available. In this case the auction has failed and the resource could start over again with a new one.

3.2 System Implementation

The original MePAge system architecture is depicted in Figure 2 and consists of three layers with different functionalities. The complexity of the layered model arises through different aspects. First, as already stated, the system has been built as simulation for benchmarking different coordination mechanisms and also as normal desktop system that could be used in field tests within a hospital. In this respect, it should be possible to reuse as much of the functional parts as possible when switching from simulation to operation. Second, MedPAge was part of a higher-level initiative called Agent.Hospital, in which several projects were integrated into a more complete hospital envi-

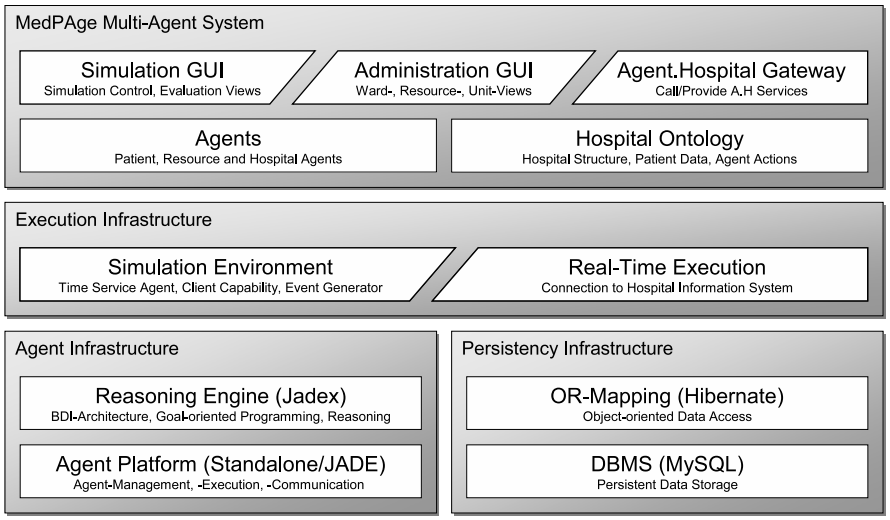

Fig. 2 System architecture

ronment. Hence, the interoperability of the system was an important aspect solved by a shared hospital ontology.

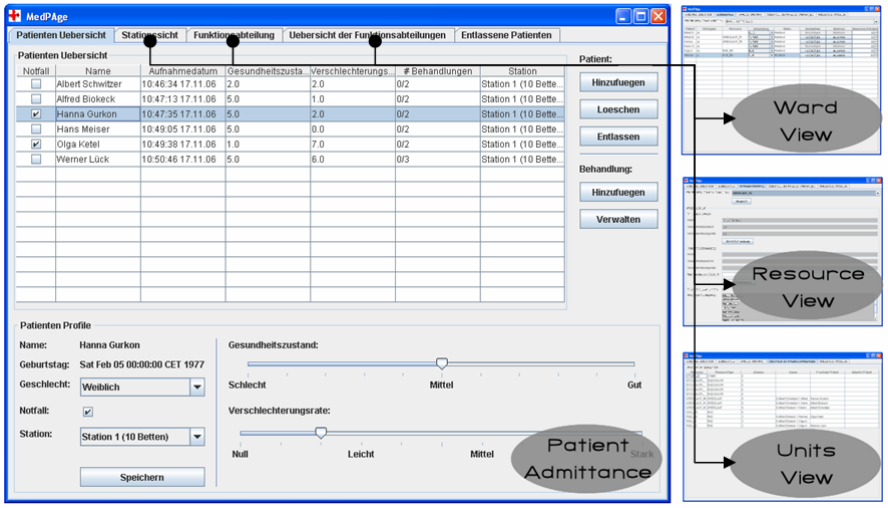

Fig. 3 System screenshot

Due to the different use cases, distinct user interfaces have been built for simulation and real world system testing. The simulation user interface allowed for performing simulation experiments and evaluating the statistical

result whereas the user interface for operation was conceived as a decision support system (cf. Figure 3). It can be seen that different views have been developed according to the functional roles involved within the hospital. From an administrative point of view the system allows for entering new patient and resource master data. This was necessary as the system has not been directly connected to the hospital server infrastructure, which would have allowed reusing existing data. Within the ward view, the current health state according to the diagnosis can be coarsely given and required treatments and examinations can be added to the patients in correspondence to the medical prescriptions. The system automatically generates appointment proposals that become visible within the unit as well as in the ward view. In the unit view the currently waiting patients are shown to the personnel and it allows for synchronizing the real hospital resource with the virtual one. For this purpose it can be entered to system when a medical task begins and ends or if a disruption exists. In the ward view the system announces treatment calls for patients according to the internally applied auction mechanism. The personnel is free to accept or reject the system proposals and send patients to the resources as they deem appropriate. These choices should be entered into the system in order to allow for correct future planning.

The core of the MedPAge system is the coordination mechanism that has been realized using a multi-agent negotiation approach. Hence, patient as well as resource agents have been modeled and implemented as representatives for their corresponding stakeholders. Details about the agent based design can be found in [4]. The coordination mechanism has been realized as message based interaction protocol in accordance to the sequence diagram shown in Figure 1. As execution infrastructure the Jadex agent platform [11] has been used, which originally relied on the JADE agent platform [1]. Hospital data and scheduling information was stored in a relational database (MySQL), which had been interfaced with an object-relational mapper (Hibernate). In order to support the simulation as well as real time execution an additional execution infrastructure middle layer had been built.

3.3 Weaknesses and Challenges

The experiences gained by implementing the MedPAge system led to several important research questions that were found to require addressing them also on a general layer and not only in a project specific manner:

- Simulation and Operation of a system should be possible without modifying the functional core of the system. Rebuilding the system from a simulation prototype leads to new failures and much additional implementation overhead.
- Implementing coordination mechanisms using asynchronous message passing is a very tedious an error prone task. Even experienced developers

struggle in foreseeing all possible protocol states leading to increased development efforts and delays in software production.

In order to address the first challenge we worked on the notion of simulation transparency [12] denoting the fact that a simulation and real system become indistinguishable from a programmers point of view (except its connection to a real or virtual environment). The underlying idea consists in introducing a clock abstraction within the infrastructure that is used for all timing purposes. Exchanging the type of clock (e.g. from a real time to an event driven mode) immediately changes the way time is interpreted in the system. Following this path made obsolete the execution infrastructure layer in case of the MedPAge system.

The second challenge requires even more fundamental changes in the way systems are designed and implemented. The proposed solution path consists in changing the communication means used in the system. On the one hand the original idea of MedPAge, namely having negotiating patients and resources should be kept, but on the other hand the communications should be simplified. To achieve this objective, a service oriented perspective is combined with the agent oriented view leading to the notion of active components described in the following Section. Relying on active components the complex interaction scheme can be largely simplified by using services. It has been shown that conceptually that agents can be equipped with services without loosing their special characteristics like autonomy [3].

4 Service-Oriented Interaction Design

In this section, a redesign of the MedPAge system is sketched that takes into account the lessons learnt from its original implementation. Here, we focus on the redesign of the coordination mechanism according to the newly developed active components approach. Therefore, the basic concepts of the active components approach are shortly presented in the next section. Afterwards, a straight forward method is proposed how to map message-oriented negotiation protocols to appropriate service interfaces. The resulting service-oriented redesign of the MedPAge coordination protocol is put forward and the advantages of the approach are discussed.

4.1 Active Components Approach

The active components approach is a unification of several successful software development paradigms, incorporating ideas from object, components, services and agents [2, 10]. Therefore, it combines features such as the autonomous execution from agents and the explicit specification of provided

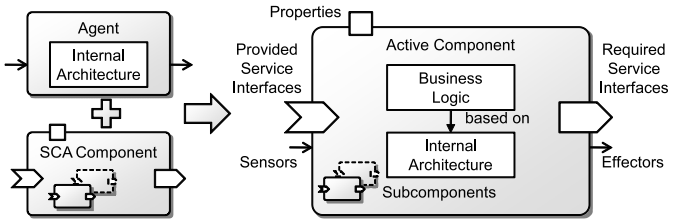

Fig. 4 Active component structure

and required interfaces from components (cf. Fig. 4). An important characteristic of the approach is that each active component may act as service provider and service user at the same time. Thus the interaction of active components is not limited to the client/server model as used in, e.g., web services, but also naturally supports peer-to-peer communication as found in multi-agent systems. Regarding the architecture of an application, the required and provided service interfaces of active components allow making component dependencies explicit and easily manageable already at design time. Yet, by supporting dynamic service search and binding, the model allows for adaptive applications that automatically build and change (parts of) their structure at runtime in response to a dynamically changing environment (e.g. when service providers disappear or new services become available).

The active components approach has many intentional similarities to the standardized and industry-supported service component architecture (SCA[1]). Yet, there are important differences due to the incorporation of agent ideas. In the active components approach, each active component is an autonomously executing entity (like an agent) that interacts with other (possibly remote) active components by calling methods of their provided service interfaces. For autonomy reasons and also for avoiding consistency issues due to concurrent access to a components internal state, active components follow a single-thread execution model per component, which means that external requests and internal actions are queued and executed sequentially (cf. [3]). To respect this execution model and also to support distribution transparency, i.e. no difference in programming for remote vs. local service calls, all service operations should use asynchronous method signatures. Asynchronous methods can be realized using so called *futures* [13] as return values, i.e. objects that act as proxies for later results of ongoing operations. Once the asynchronously invoked operation finishes, the actual result will be made available in the future object.

In Jadex[2], which is our Java-based reference implementation of the active components approach, different kinds of futures are provided that capture recurring interaction patterns in a natural way. In the following, three im-

[1] `http://www.oasis-opencsa.org/`

[2] `http://jadex-agents.informatik.uni-hamburg.de`

portant types are introduced by dint of a simple, illustrative example of a wearable heart rate monitor connected wirelessly to a stationary display device. The heart monitor service allows retrieving the last observed heart rate and also creating an electrocardiogram (ECG) plot of the historic heart activity according to a specifiable period. In addition, the display can subscribe to the heart rate monitor for being sent periodic updates of the current heart rate. As shown in Fig. 5 (line 2), the *getHeartRate()* method of the monitoring service returns a simple future of type double. Because the service execution is done asynchronously, the rate value is not immediately available. Instead, the display component may add a listener to the future (line 3) for being informed, when the service execution is finished and the result can be obtained. For long running operations, the terminable future as used in the *generateECGPlot()* method (lines 4, 5) allows terminating an ongoing operation, while it is still in progress (cf. line 7). This kind of interaction is useful, when service calls may take a long time to complete and allow, e.g., the user to abort the requested printing of the ECG plot and choose a shorter period. Finally, for recurrent interactions, the intermediate future type allows the service implementer to publish intermediate results of potentially long-lasting operations. A special kind of intermediate future is the subscription future, which resembles a publish/subscribe interaction. For the *subscribeToHeartRates()* method (line 9, 10) you can see that the display is informed about each new heart rate value in the *intermediateResultAvailable()* method. To cancel such a subscription, also the *terminate()* method of the future can be used, i.e. every subscription future is also an instance of the terminable future type.

```
01: IHeartMonitorService heartservice;
02: IFuture<Double> rate = heartservice.getHeartRate();
03: rate.addResultListener(...);

04: ITerminableFuture<ECGPlot> generateplot
05:   = heartservice.generateECGPlot(long period);
06: ...
07: generateplot.terminate();

08: ISubscriptionIntermediateFuture<Double> heartrates
09:   = heartservice.subscribeToHeartRates();
10: heartrates.addResultListener(new IntermediateResultListener() {
11:   public void intermediateResultAvailable(Decimal rate) {
12:     // update display value...
13:   }
14: })
```

Fig. 5 Examples of future types

4.2 Modelling Interactions with Services

In message-based interactions, each communicative act is represented as a simple message. As inspired by speech-act theory, difference between those acts can be identified by using performatives such as *query* or *inform*. The allowed sequences of sent and received messages can be specified in interaction protocols, which can be specified, e.g., in AUML sequence diagrams. The foundation for intelligent physical agents (FIPA) has proposed a standardized set of performatives and also defined several commonly used interaction protocols like *request* and *contract-net*. Despite many attempts for providing tools or other development support, the current practice of implementing of message-based interactions still exhibits a large gap between design and the actual programming. Message-based interaction is not naturally supported by today's prevalent programming languages like Java and C#. This requires implementing many aspects of the interaction hidden in application code. As a result, implementation errors, such as type errors or invalid message sequences, can only be detected at runtime, if at all. This makes implementing message-based interactions a tedious and error-prone, time consuming task.

On the other hand, for the communicative acts in service interactions, different representations exist for, e.g., a service call, its return value or potential exceptions. The future patterns introduced above allow further types of communicative acts, such as termination of ongoing operations or publication of intermediate results. All these types of interaction are naturally represented by well-established object-oriented concepts like interfaces, method signatures and futures. Because these constructs are all first class entities of current object oriented programming languages, sophisticated tool support exists for statically checking the soundness of interaction implementations already at compile time (e.g. checking number and type of parameters, compatibility of return values or the types of futures supported for an interaction).

For the above reasons, we consider a service-based implementation of an interaction advantageous to a message-based one. Still the design of a service-based interaction is a non-trivial task, because in addition to deciding about sequences of communicative acts, also each act needs to be mapped to one of many fundamentally different interaction types, such as service calls, return values, etc. Therefore we propose an approach starting from a message-based interaction design and converting it to a service-based design for easier implementation. In this way the decisions about which communicative acts are required and how these are represented are straightened, thus simplifying the design process. In addition, the design of message-based interactions is well-understood and it seems reasonable to reuse the existing concepts, standards and tools from this area.

As a result from practical experiences in converting message-based designs to service-based ones, we identified the following recurrent steps:

1. *Identification of service provider:* In almost every case, the initiator of an interaction acts as client of a participant, i.e. the first communicative act of an interaction is best represented as a service call from the initiator to the corresponding participant.
2. *Mapping of subsequent communicative acts:* For each subsequent communicative acts it needs to be decided how it is represented according to the following options:

   a. Messages from initiator to participant can be represented as service calls or as termination request to previous service calls.
   b. Messages from participant to initiator can be represented as service result, intermediate service result or as failure (exception) of a previous service call.

3. *Evaluation of the resulting design:* The design should be evaluated according to the following design criteria

   a. Establishing high cohesion of methods in any interface and low coupling between different interfaces.
   b. Minimizing dependencies between methods (e.g. avoid that methods are only allowed to be called in a special ordering).
   c. Avoiding to introduce new communicative acts (e.g. when an act is mapped to a service call, but there is no message corresponding to the return value).

4. *Splitting Interfaces:* If no appropriate design can be found with a single service interface, the interaction needs to be split up into several interfaces. Therefore one or more messages in the interaction need to be chosen as start of a sub-interactions and the process needs to be repeated from step 1 for each sub-interaction.

Of course, in practice these steps should be considered as guidelines and not as a rigid process. For each concrete interaction, probably different designs need to be considered to identify advantages and limitations and to decide on the best option for implementation.

4.3 Redesign of the MedPAge Coordination Mechanism

The service-based redesign process has been applied to the MedPAge coordination mechanism as presented in Section 3.1. The resulting interfaces are shown in Fig. 6. The design has been obtained by following the process outlined in the previous section. The interaction is initiated by a patient, which has some treatments without a corresponding reservation at a resource, yet. Therefore the resource is the participant of the interaction and should provide a service interface to the patient (lines 1-5). During the redesign it became

obvious that some of the messages of the original interaction are obsolete, especially in the announcement phase. Therefore these messages were omitted during the redesign. In the original interaction protocol, the resource would periodically start auctions in which the registered patients could bid for an upcoming time slot. This naturally maps to a service call by the patient to register a required treatment and in response the resource would provide a subscription future to publish the start of each auction (lines 2, 3). In order to simplify the interaction and also to allow for dynamic changes in the resource operation, it was decided, that the stochastic treatment durations are supplied by the resource to the patient at the start of each auction. Therefore the *subscribeToAuctions()* method of the resource service corresponds to the initial *request(treatment)* message. It should be noted that the Jadex active components infrastructure automatically makes available the caller of a service to the service implementation and thus no separate parameter for identifying the patient that called the resource method is necessary. The subscription future returned by the method represents both the *inform(durations)* message as well as the *cfp(treatment)* message. To include information about the auctioneered time slot and the corresponding treatment durations, the *CFP* object type is introduced (lines 9-14) and used as result type of the subscription future.

```
01: public interface IResourceService {
02:   public ISubscriptionIntermediateFuture<CFP>
03:     subscribeToAuctions(String treatmenttype);
04:   public ITerminableFuture<Boolean> bid(CFP cfp, double costs);
05: }

06: public interface IPatientService {
07:   public IFuture<Boolean> callPatient();
08: }

09: public class CFP {
10:   public String treatmenttype;
11:   public long startdate;
12:   public double duration;
13:   public double variance;
14: }
```

Fig. 6 Redesigned MedPAge coordination mechnism

When a new auction starts, the patient receives this CFP object calculates its opportunity costs and sends its bid to the resource. This is done using the *bid()* method (line 4) corresponding to the *propose(bid)* message. The result of the method can be interpreted as corresponding to the *query-if(accept-perform)* and *reject(bid)* message in case of success or failure. Yet, this would

leave the three remaining messages *refuse(not-available)*, *cancel(treatment)* and *agree(perform)* to be realized by an isolated method in the resource interface. This would have violated design criteria 3b (minimizing dependencies), because the required ordering of the three resource interface methods would not have been obvious from their syntactic structure. Instead it was decided, that the majority of the awarding phase was better represented by a separate patient interface (lines 6-8). Here the resource can inform the winner of an auction using the *callPatient()* method corresponding to the *query-if(accept-perform)* message. The patient may answer with the boolean result value if it is available or if the resource should prefer another patient. The advantage of this design is that the call-patient functionality can also be used independently of the auction mechanism. E.g. a resource that wants to apply a different scheduling scheme may simply accept registrations in the *subscribeToAuctions()* method, but never start any auction and instead call patients directly as deemed appropriate.

4.4 Discussion

In the following, the specific improvements of the MedPAge coordination algorithm are discussed. In the original message-based MedPAge coordination protocol, twelve different messages between two roles have been used. Without the obsolete announcement phase still nine messages remain. The redesigned service-oriented coordination mechanism captures the same functionality, but only requires a total of three methods spread over two interfaces. Thus, the perceived complexity in the service-based interaction design is much lower than in the message-based design. Furthermore, the object-oriented service interfaces are conceptually much closer to the implementation technology, which even more reduces the complexity for the programmer. On a conceptual level, the new design keeps the agent-oriented view of patient and resource representatives, which negotiate for finding bilaterally appropriate schedules. Therefore, the autonomy of the different hospital units is preserved. Yet, the object-oriented design makes dependencies between the communication acts also syntactically clear. E.g., the *subscribeToAuctions()* method needs to be called before the *bid()* method, because the latter requires the *CFP* object supplied by the former method.

5 Conclusion

In this paper patient scheduling has been identified as important area of improvement in hospitals. The currently employed technique for patient scheduling is a first-come first-served scheme that has the advantages of being highly

flexible and respecting the autonomy of the hospital units. Within the MedPAge project a decentralized auction-based solution has been developed, which preserves these characteristics but additionally adds a global optimization perspective. The MedPAge system has been built as an agent-oriented solution. Based on the original system design and implementation some of its weaknesses and resulting challenges have been identified. The active components approach and in particular the service-oriented interaction design have been presented. They address the weakness of the complexity and implementation effort that is induced by message-based interaction designs. A service-oriented redesign of the MedPAge coordination mechanism has been developed and its advantages have been discussed.

During the course of the MedPAge project, the agent metaphor was found a highly suitable design approach for decision support systems in the area of hospital logistics. The agents can act as representatives of the involved hospital stakeholders, taking into account their respective goals and the typical local autonomy of existing hospital structures. One remaining obstacle for putting agent-based systems into productive use is still the gap between agent technology and established mainstream technologies, such as object-orientation and, e.g., web services. The service-based approach presented in this paper supports an easier integration with legacy software, allowing, e.g., to use web services for provided or required service calls.

References

1. Bellifemine, F., Caire, G., Greenwood, D.: Developing Multi-Agent systems with JADE. John Wiley & Sons (2007)
2. Braubach, L., Pokahr, A.: Addressing challenges of distributed systems using active components. In: F. Brazier, K. Nieuwenhuis, G. Pavlin, M. Warnier, C. Badica (eds.) Intelligent Distributed Computing V - Proceedings of the 5th International Symposium on Intelligent Distributed Computing (IDC 2011), pp. 141–151. Springer (2011)
3. Braubach, L., Pokahr, A.: Method calls not considered harmful for agent interactions. International Transactions on Systems Science and Applications (ITSSA) **1/2**(7), 51–69 (2011)
4. Braubach, L., Pokahr, A., Lamersdorf, W.: MedPAge: Rationale Agenten zur Patientensteuerung. Künstliche Intelligenz (2), 33–36 (2004)
5. Dept of Veteran's Affairs, Australia: Clinical Pathway Manual for Geriatric Community Nursing (2000). `www.dva.gov.au/health/provider/provider.htm`
6. Paulussen, T.: Agent-Based Patient Scheduling in Hospitals. Ph.D. thesis, Universität Mannheim (2005)
7. Paulussen, T., Zöller, A., Rothlauf, F., Heinzl, A., Braubach, L., Pokahr, A., Lamersdorf, W.: Agent-based patient scheduling in hospitals. In: P.L.O.S. S. Kirn O. Herzog (ed.) Multiagent Engineering - Theory and Applications in Enterprises, pp. 255–275. Springer (2006)
8. Paulussen, T.O., Heinzl, A., Rothlauf, F.: Konzeption eines koordinationsmechanismus zur dezentralen ablaufplanung in medizinischen behandlungspfaden (medpaco). In: 5. Internationale Tagung Wirtschaftsinformatik 2001, pp. 867–880. Physica, Heidelberg (2001)

9. Paulussen, T.O., Zöller, A., Heinzl, A., Pokahr, A., Braubach, L., Lamersdorf, W.: Dynamic Patient Scheduling in Hospitals. In: M. Bichler, C. Holtmann, S. Kirn, J. Müller, C. Weinhardt (eds.) Coordination and Agent Technology in Value Networks. GITO, Berlin (2004)
10. Pokahr, A., Braubach, L.: Active Components: A Software Paradigm for Distributed Systems. In: Proceedings of the 2011 IEEE/WIC/ACM International Conference on Intelligent Agent Technology (IAT 2011). IEEE Computer Society (2011)
11. Pokahr, A., Braubach, L., Lamersdorf, W.: Jadex: A BDI Reasoning Engine. In: R. Bordini, M. Dastani, J. Dix, A. El Fallah Seghrouchni (eds.) Multi-Agent Programming: Languages, Platforms and Applications, pp. 149–174. Springer (2005)
12. Pokahr, A., Braubach, L., Sudeikat, J., Renz, W., Lamersdorf, W.: Simulation and implementation of logistics systems based on agent technology. In: T. Blecker, W. Kersten, C. Gertz (eds.) Hamburg International Conference on Logistics (HICL'08): Logistics Networks and Nodes, pp. 291–308. Erich Schmidt Verlag (2008)
13. Sutter, H., Larus, J.: Software and the concurrency revolution. ACM Queue **3**(7), 54–62 (2005)
14. Zöller, A., Braubach, L., Pokahr, A., F. Rothlauf, T.P., Lamersdorf, W., Heinzl, A.: Evaluation of a multi-agent system for hospital patient scheduling. International Transactions on Systems Science and Applications (ITSSA) **1**, 375–380 (2006)