Integrating Web Services in Petri Net-based Agent Applications

Tobias Betz, Lawrence Cabac, Michael Duvigneau, Thomas Wagner, Matthias Wester-Ebbinghaus

University of Hamburg
Faculty of Mathematics, Informatics and Natural Sciences
Department of Informatics
{betz,cabac,duvigne,wagner,wester}@informatik.uni-hamburg.de
http://www.informatik.uni-hamburg.de/TGI

Abstract. The context of this paper is given through a software engineering approach that uses Petri nets as executable code. We apply the particular understanding that Petri nets are not only used to model systems for design purposes but also to implement system components. Following this approach, we develop complex Petri net-based software applications according to the multi-agent paradigm. Agent-internal as well as agent-spanning processes are implemented directly as (high-level) Petri nets. These nets are essential parts of the resulting software application – alongside other parts (operational and declarative ones), which are implemented using traditional ways of programming.
One of our goals is to open our Petri net-based agent framework Mulan/Capa so that multi-agent applications can communicate and interoperate with other systems – especially with Web-based applications. For this cause, we present a gateway solution to enable Petri net-based applications to access Web services as well as to offer Web services to other applications: the *WebGateway*. Besides describing the WebGateway extension itself, we use its presentation to demonstrate the practicability of the Petri net-based software engineering approach in general. We emphasize two benefits: (1) Petri net models serve as conceptual models that progressively refine the constructed system from simple models to well-defined specifications of the systems. This improves the understanding of the systems. (2) Having essential parts of the software system being implemented with Petri nets allows to carry out (partial) verification of our application code by means of standard formal methods from the field of Petri net theory.

Keywords: Web Services, High-Level Petri Nets, Multi-Agent Systems, Mulan, Renew, P*aose

1 Introduction

One of the most frequent requirements for modern software applications is to open the access of the offered functionality to other entities in the World Wide

Web. In this paper we address the topic of meeting this requirement for Petri-net based software applications. We present a gateway solution for allowing Petri net-based applications to access Web services as well as offering Web services themselves.

The usefulness of Petri nets for software engineering has been recognized in the context of many paradigms like object-orientation [1], components / plugins [7,16,22] or agent-orientation [17,20]. However, in this paper we assume a particular understanding of *Petri net-based software*. In addition to using Petri nets for design-level artifacts and for verification of certain system properties, we utilize Petri nets as our *implementation language*. More concretely, we rely on the high-level Petri net formalism of *Java reference nets* [7,18] that allows to combine multi-level Petri net modeling (according to the *nets-within-nets* concept [25]) with Java programming. The formalism is supported by the Renew[1] tool (`http://www.renew.de`). We have developed the multi-agent system (MAS) framework Mulan[1] based on Java reference nets. It provides a powerful middleware for running distributed multi-agent applications on multiple instances of Renew. In addition, we have developed a Petri net-based agent-oriented software engineering approach (P*aose[1]) for the construction of such multi-agent applications.

With the *WebGateway* extension, we introduce the latest addition to our Petri net-based software engineering framework Mulan/Capa. While the Mulan model is often referred to as the reference architecture, Capa (Concurrent Agent Platform Architecture) is an extension and implementation of Mulan. Capa [11] provides convenient ontology-based message processing and an infrastructure for FIPA-compliant agent management and IP-based transport services. Capa is, thus, one implementation of the reference architecture Mulan. It allows to integrate Mulan applications into Web-based environments via Web services. This *opens* Mulan applications in the sense that Mulan agents can now access resources external to the agent world in a uniform way via Web Services instead of having to be equipped with proprietary connectors. In the other direction, Mulan agents publish and offer their own services also as Web services and thus can equally be accessed uniformly from anywhere across the Web. Again, we stress our specific understanding of Petri net-based software, which carries over to the integration with Web services. Usually, work on Petri nets and Web services deals with providing semantics to modeling notations that are used within the Web context, like BPEL [13] and some translations [15]. Our approach is to provide a way to offer Web services that are actually *realized by the execution of Petri net models*. The other way round, the *execution of Petri net-based applications can include the access of arbitrary Web services*.

The concrete aim of this paper is twofold. Firstly, we introduce the WebGateway itself as a solution for bringing Petri net-based applications and Web services together. Secondly, we use the WebGateway extension as an example

[1] For more detailed information about Renew, Mulan and P*aose see [5] and `http://www.paose.net`.

for the general benefits that underlie our approach of Petri net-based software engineering. We claim that these benefits are basically:

1. We follow an engineering approach, in which we move from conceptual models as design artifacts to refined, technical models as software artifacts. In our opinion, this approach of *implementation through specification* allows to iteratively build models/code in a documented and comprehensible way. In addition, core features of a system can be determined early on and maintained in further refinements.
2. By using *Petri nets as code* we can verify our application code. Of course, this can only happen within certain limits. Both the nets-within-nets nature of our models and the use of Java inscriptions prohibit a comprehensive verification. Nevertheless, we can define abstractions (e.g. P/T net abstractions) of our program code and verify specific properties (e.g. properties of sound workflows) with respect to them. This allows at least partial verification of our application code (which could possibly be supplemented with unit testing mechanisms for reference nets, cf. Section 8 and [8]).

The outline of the paper is as follows. In Section 2, we present the conceptual model of our WebGateway extension for bringing Petri net-based (agent-oriented) applications and Web services together. Based on this, we present details of the WebGateway implementation in Section 3. In combination, these two sections demonstrate the benefits and practicability of our *implementation through specification* approach. In Section 4, we demonstrate how the WebGateway and one particular Petri net-based Web service are deployed in the context of our integrated project management environment (IPME) on a day-to-day basis. Section 5 presents a further insight to the implementation in order to demonstrate the refinement process of the model. We discuss the results of the paper in Section 6, position them in the context of related work in Section 7 before we conclude the paper in Section 8.

2 Conceptual Gateway Architecture

The WebGateway extension to our multi-agent framework Mulan/Capa is realized by a *WebGateway agent*. This agent is coupled to a (jetty) Web Server and thus brings the two worlds of the (Web service-based) Internet and multi-agent systems together. In Section 3, we address technical details of the WebGateway realization. In this section, we focus on the conceptual architecture. Along the way, we demonstrate the usefulness of applying a Petri net-based modeling approach. We progressively refine a simple architectural model to a meaningful and well-defined specification. This specification then represents the basis on which to actually implement the WebGateway agent's behavior. Due to space limitations, we cannot address the whole functionality of the WebGateway. Instead we concentrate on handling Web requests and responses. Further topics and challenges will be addressed in the following section.

Figure 1 illustrates our starting point. In order to provide communication in such a heterogeneous setup consisting of Web and multi-agent system parts, the communication has to be facilitated. The WebGateway provides the translation as an adapter between the two worlds of communication. For this it provides two interfaces: one for Web-based communication and one for FIPA[2]-compliant communication. The interfaces are depicted in Figure 1 as white rectangles. The Internet is shown as a cloud and the MULAN reference model (cf. [17]) stands as an exemplary multi-agent system.

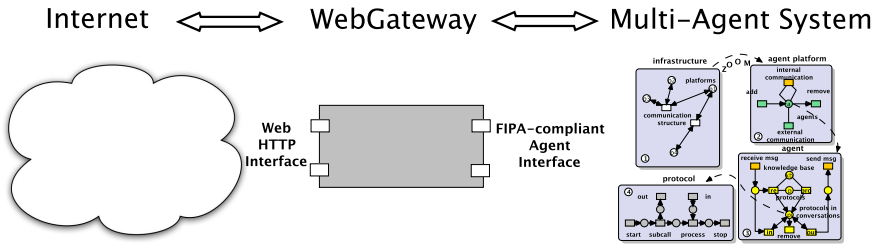

Figure 1. WebGateway context.

In the following we will consecutively refine the WebGateway as a Petri net model. Figure 2 shows the first step of this refinement. The WebGateway's main functionality of translating messages from one domain to the other is represented as two transitions that are included in the transformation component. The two interfaces are now depicted as transitions.[3] Messages may enter through the interface transitions with outgoing arcs, are received on the buffer places and ready for transformation processing.

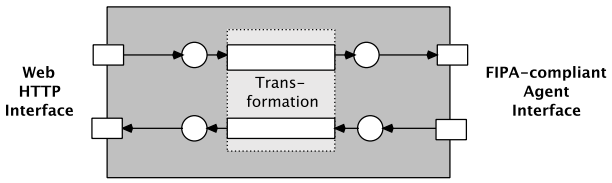

Figure 2. Simple WebGateway architecture model

[2] Foundation for Intelligent Physical Agents: `http://www.fipa.org`.

[3] The notion of transitions being interfaces fits nicely with an object-oriented paradigm, if one presumes that these (on one side open) transitions are one port of synchronous channels.

Translation is an important and already technically challenging part (cf. the following section) but by far not the only task of the WebGateway. In addition, it has to make sure that responses are matched to requests across the heterogeneous setup. In our approach the WebGateway keeps a copy of a request message in order to be able to provide this matching. Thus responses can be routed to the right recipient. In Figure 3 an exemplary conversation direction is modeled: a request from the Web to an agent-implemented service. The original request (e.g. from a Web browser) is a JSON message (JavaScript Object Notation). The WebGateway translates this message to FIPA-SL (Semantic Language), which can be understood by agents in the multi-agent system. A copy of the message is kept within the gateway, which allows the gateway to route the response message to the requester after the answer has been translated back from FIPA-SL to JSON. Please note that JSON stands for just one possibility of a Web message's content. Content types like XML or HTML form data can be translated in a similar fashion.

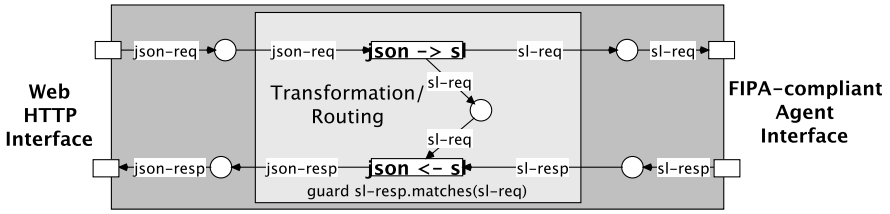

Figure 3. WebGateway architecture model for handling Web-client requests

While Figure 3 covers one exemplary interaction type, namely a request sent by a Web-based client to an agent-based service, the WebGateway also provides the possibility that a Web service request is initiated from the agent's side. In this case an SL encoded request will be translated to (for instance) JSON, a copy of this message will be kept for later routing and the answer from a Web application will be translated from JSON to SL in order to deliver it to the requesting agent. Both initiating directions are supported by the WebGateway and use the same interface as depicted in Figure 4.

In addition to the request interactions covered so far, the WebGateway also supports a uni-directional communication (*inform* interaction), which is not discussed here.

While this conceptual Petri net model of the WebGateway architecture shows the basic (internal) behavior of the WebGateway, it neither specifies the interactions between WebGateway and Web applications or agents nor does it present a realistic level of detail for the implementation of the WebGateway agent. These details are covered in the following section.

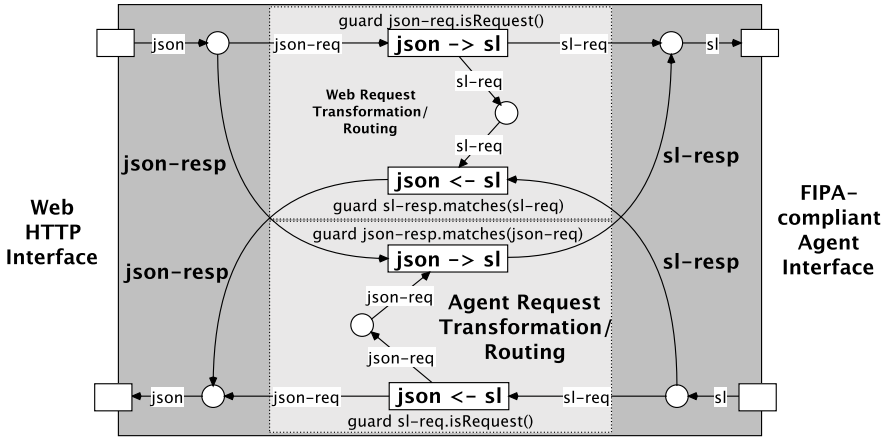

Figure 4. WebGateway architecture for two-way service request handling

3 WebGateway: Integration and Details

In order to achieve a concrete implementation of the abstract architecture described in the previous section we need to combine multiple technologies, which are well established in the world of multi-agent systems and Web services. In this section, we describe how those technologies are combined for the implementation of the WebGateway in order to obtain the desired integration of both application domains. For this we present two parts of the adapter functionality of the WebGateway. The first is concerned with the message routing and translation as well as service registration. It focuses on the Web interface side, which is – from the perspective of the muulti-agent system – the external interface. The realization of this interface, which requires the integration of the required technologies, is presented in Section 3.1. The second part focuses on the WebGateway as a part of the agent system and its communication with other agents. Hence, Section 3.2 introduces the communication protocols providegd with the WebGateway.

3.1 Integration of the Required Technologies

An initial requirement is that the WebGateway must be able to interact with communication partners of both worlds. For that reason the WebGateway provides two communication interfaces as shown in Figure 5 where we have included the conceptual architecture model from the previous section in order to illustrate its relation to the actual WebGateway implementation.

As the WebGateway is realized as an agent itself, the agent interface for communication with other agents is inherently part of the underlying multi-agent system framework (Mulan/Capa in our case). Consequently, the *ordinary* FIPA-compliant agent communication infrastructure of our framework enables

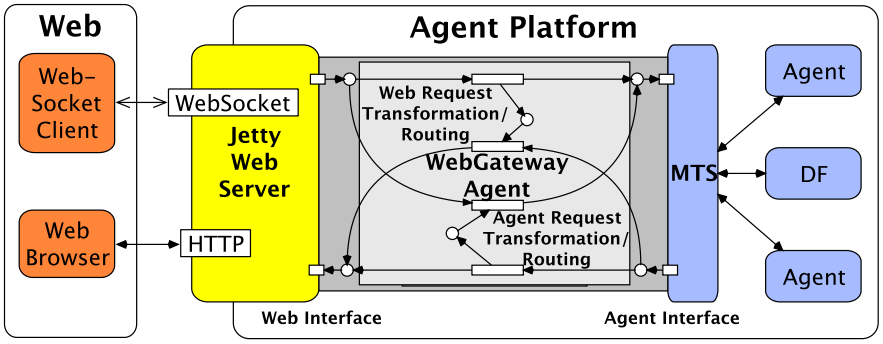

Figure 5. Integration of the WebGateway in the multi-agent system

communication with other agents both on the same platform and on remote platforms. This part can be considered as the WebGateway's *internal communication interface*.

In addition, the WebGateway agent needs a Web interface that serves the communication with Web services and Web clients. It is realized using a Web server (Jetty, `http://www.eclipse.org/jetty`). This is where we have extended our framework. For each MULAN/CAPA host, one Web server is launched. A MULAN/CAPA host may include multiple agent platforms, but typically we have a one-to-one correspondence between a host and a platform. For each platform, a WebGateway agent is launched and connects automatically to the platform's (/host's) Web server. This part can be considered as the WebGateway's *external communication interface*. The Web server enables communication between the WebGateway agent and Web services/clients using the well established HTTP protocol (Hypertext Transfer Protocol, `http://www.w3.org/Protocols`) as well as the HTML5 WebSocket protocol (`http://dev.w3.org/html5/websockets/`). In contrast to HTTP, a WebSocket connection allows to exchange messages asynchronously between client and server. This allows more flexibility for browser-based Web applications and fits quite well with the agent paradigm as it traditionally relies on asynchronous interactions.

Besides mediating communication *technically*, there remain further challenges in order for the WebGateway to really provide a transparent and bidirectional communication between the agent and the Web service world. We identify the following required key features.

1. Routing and management of messages between the different interfaces.
2. Registration and management of agent services that are published as Web services and vice versa.
3. A two-way translation of the supported message encodings.

The first mentioned feature is specifically addressed by the conceptual architecture for the WebGateway from the previous section. More (technical) details

of our solution concerning the cross-technological tracking and routing of messages can be found in [4].

For the second mentioned feature, our approach supports – and actually is limited to – RESTful Web services. The **RE**presentational **S**tate **T**ransfer (REST) architecture [12] gained increased attention because of its simplicity of publishing and consuming services over the Web. The architecture is based on resources that identify the participants of service interactions and that are addressed with globally unique URIs[4]. Such a resource can be manipulated over a uniform interface with a fixed set of operations (GET, POST, PUT, DELETE, etc.) that are traditionally part of the underlying HTTP networking protocol. Resources are also decoupled from their representations so that their content can be accessed in a variety of formats e.g. HTML, JSON[5], XML or even JPEG images. For our purposes, we treat artifacts from the multi-agent world (hosts, platforms, agents, agent services) as REST resources in the Web world. The technical counterparts for these *agent-based REST resources* on the Web server side are implemented as Java Servlets[6]. These are responsible for providing the resource representations and also act as connection endpoints for HTTP and WebSocket connections, forwarding all incoming messages to the responsible WebGateway agent. In [4], we provide more details on addressing agent-based REST resources and on how to provide suitable presentations.

The last mentioned feature was also briefly addressed in the previous section (translation between FIPA-SL and JSON/XML/HTML form data). We will not cover this topic here, but again refer to [4].

3.2 The WebGateway as a Mulan/Capa Agent

So far we have mainly focused on Web technologies that are necessary for realizing the WebGateway according to the conceptual architecture presented in the previous section. However, we have already stressed the fact that the WebGateway is actually realized as an *agent* in our Mulan/Capa framework. We have presented the development approach (P*aose) for Mulan/Capa multi-agent systems together with corresponding tools on other occasions (cf. [5,6]). Basically, a Mulan/Capa agent is designed in terms of three aspects: agent knowledge, agent-internal processes, agent-spanning processes. These aspects eventually manifest in three types of software artifacts for agent implementation: a *knowledge base*, *decision components* (DCs) for managing agent-internal processes and *protocols* for managing interactions with other agents. In this paper, we will not comprehensively cover all details of developing the WebGateway agent but provide an overview of the necessary parts.

Basically, the conceptual architecture described in Section 2 provides the groundwork, on which the agent's decision components are designed. In the previous subsection, we have covered the technologies that are needed to flesh out the conceptual architecture in order to arrive at an actual implementation.

[4] Uniform Resource Identifier
[5] Javascript Object Notation (JSON)
[6] `http://www.oracle.com/technetwork/java/index-jsp-135475.html`

One central aspect of agent design is its interactions with other system parts. Interactions between the WebGateway and Web applications take place via HTTP/WebSockets. For interactions between the WebGateway and other agents we have to provide equally well-defined protocols. Basically, the WebGateway agent offers five protocols for this purpose:

- WebGateway_registerAgent for registering agent services as Web services.
- WebGateway_sendrequest for forwarding request from Web applications to application agents
- WebGateway_receiverequest for forwarding request from application agents to Web applications
- WebGateway_sendinform and WebGateway_receiveinform for sending inform messages in both directions

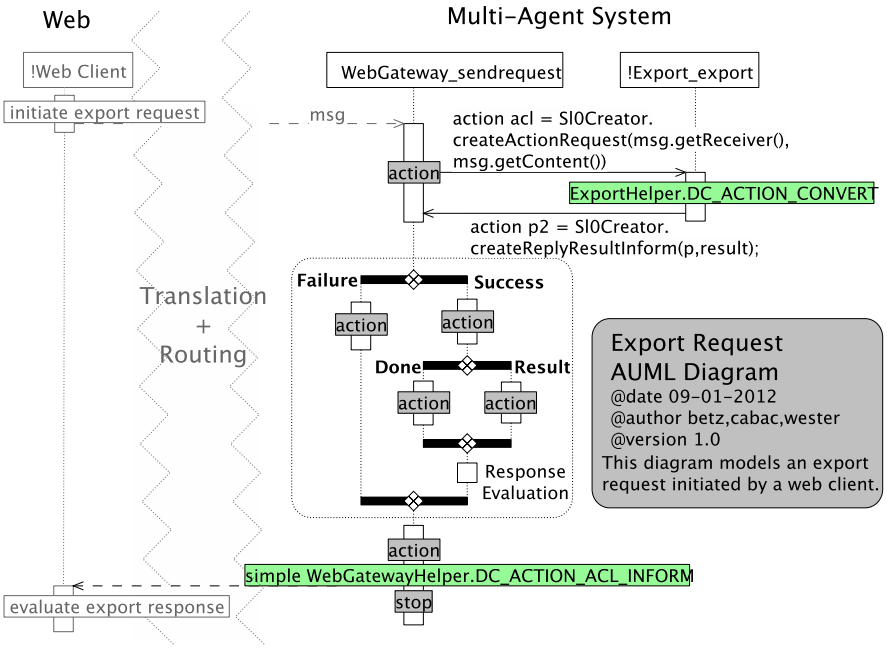

Figure 6. A model of the export request interaction initiated by a web client.

We cannot cover all of these interactions but will turn to one example. Figure 6 shows an AUML diagram for the case of a WebGateway_sendrequest. The AUML (Agent UML, see [9,6]) Interaction Protocol diagrams are derived from Sequence Diagrams. They allow to fold several sequences into one scenario by providing modeling elements for alternatives or concurrency – similar to UML2 Interaction Diagrams. Here, we regard the sample case where a Web application

requests the export functionality of an application agent and the WebGateway agent acts as the mediator. We address the export application scenario more deeply in the next section.

In the P*AOSE approach, we use these AUML diagrams to (semi-)automatically generate the interaction parts for each party as Petri nets (cf. [9]). These resulting *protocol nets* are then directly used for the implementation of agent interactions. It is important to note that the WebGateway_sendrequest protocol net is generic. Here, it is shown in the context of the export example. But is designed to be applicable for arbitrary requests sent from a Web application to an application agent. For instance, we have developed a web component-based GUI framework for browser applications that relies on exchanging web component events between browsers and agents (so called *Agentlets*).

For each new way of making use of the WebRequest_sendrequest protocol net, the two interaction sides have to *fit* together. This means that the composition of the WebRequest_sendrequest protocol net and its counterpart protocol net has (at least) to result in a sound protocol.[7]

4 Application of the WebGateway / Export Example

As a real world example we present an application that utilizes the WebGateway's functionality to provide a Web service. The *Export Service* takes a representation of a Petri net in the form of a serialized Renew drawing or as PNML and returns an image representation of the model.[8] The *Diff Service* service takes two representations of models and returns a graphical diff [10] of the two models.

People can access these Web services directly through a Web page interface. But the main application for the Web services is currently a different one. In the context of software engineering, in which we use our models, a tight integration of available tools ensures the efficiency and therefore the acceptance of the available tools among developers. Thus we have integrated the two Web services in our preferred integrated project management environment (IPME, see [3]).

Figure 7 shows a schematic model of the setup of our IPME. The IPME – in this case: *Redmine* (`http://www.redmine.org/`) – runs on a standard Web server shown in the center of the model. It includes several plugins for the access of the source code management system (SCM, possibly located on another server) and the Export/Diff Web services (again located on another server). Developers can interact with the source code repositories to introduce new versions

[7] Although the soundness property is not well-defined for protocol nets, we conceive this property in analogy to soundness of workflow nets.

[8] In fact the service takes any file type that can be read by Renew, e.g. Renew nets (.rnw), JHotDraw drawings (.draw), PNML, several diagram types used within P*AOSE (.aip,.arm) and also Lola net files (.net). The RENEW import/export system is also extensible, so any envisioned file type in the context of Petri nets and UML modeling can easily be implemented.

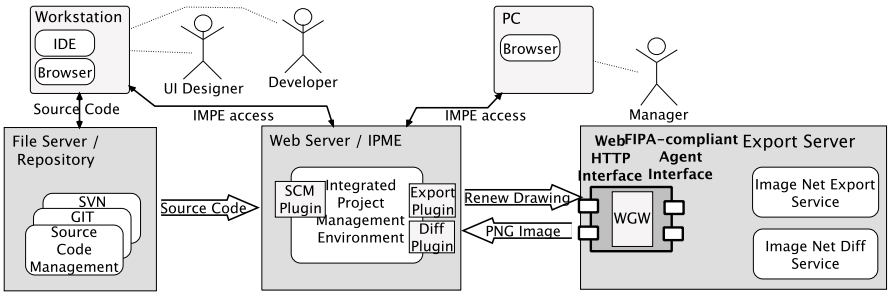

Figure 7. A schematic architecture for the export Web Service / Redmine plugin.

of artifacts, to examine the commit history or to examine differences of the selected versions and so on. Managers as well as developers can in addition use the IPME – besides of using the planning and documentation features – to investigate the source repository comfortably in a Web browser. One main part of the functionality provided by the IPME is that developers and project managers can browse quickly through the source code and choose to display a diff of versions of the source code in a Web browser. However, the default browsing and diffing functionality of IPMEs works on text-based source code only while a large part of our code base is Petri net-based. A textual representation of diagrams – for example in PNML – is not very significant for human readers. Moreover, a text based diff of versions of the diagram's text representations is completely useless. Thus, the Export and Diff plugins take the text representations for diagrams from the SCM, hand them to the Export and Diff Web services and integrate the returned images smoothly into the Web page-based display for the developer. Figure 8 shows a screenshot of the integration of the Diff Service in the IPME Redmine. The screenshot shows The Redmine user interface in a Web browser. The diff of revisions 9044 and 9545 of the *Receiver_chat* protocol net is displayed. The differences are highlighted in red (removals) and green (additions).[9] All other graphical elements are faded to a foggy gray leaving a shadow of the original net.

Consequently, within our development environment the Web services are used by the IPMEs[10] and are thus provided to the developers in an automated way. A server instance of Renew is running and provides the Web services using the Mulan/Capa framework with its WebGateway extension. A publicly accessible Export/Diff Web page and a demonstration page of the Redmine integration can be accessed from the P*aose Web Site (`http://www.paose.net/`).

Figure 9 shows a screenshot of the presented multi-agent application showing an export interaction. The MulanViewer on the left shows the multi-agent system's status in terms of all started agents, their decision components, knowledge

[9] In black & white printing the location of the manipulated parts are still recognizable, although it becomes impossible to distinguish removals from additions.

[10] We provide plugins for Redmine and Trac (`http://trac.edgewall.org/`).

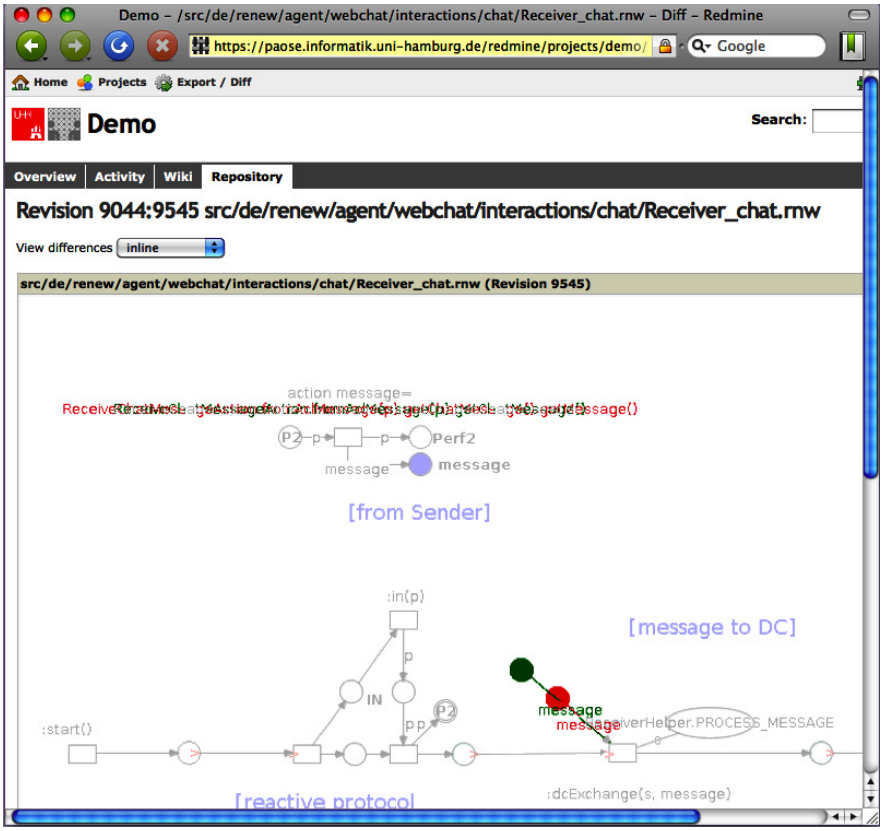

Figure 8. Screenshot of a diff image integrated in Redmine (Demo).

bases and currently executed protocol nets. In the back, parts of the involved nets are shown: these are – from top to bottom – the *transformation* decision component of the WebGateway and the *export* protocol net of the Export agent. The *sendrequest* protocol net of the WebGateway is not shown but listed in the MulanViewer's tree view. During this interaction the WebGateway *sends* the request to the Export agent after the request has been *received* from the Web client. On the right hand side of the screenshot are two frames showing a deep inspection of tokens, which are located as indicated on the highlighted places. The first one shows the *request* message in FIPA-SL, waiting to be matched with the response for routing purposes (cf. Section 2). The second is the response message, which is just about to be sent from the Export agent to the WebGateway.[11]

[11] Although the messages could also be inspected in String representation, the UML representation is much clearer and more concise.

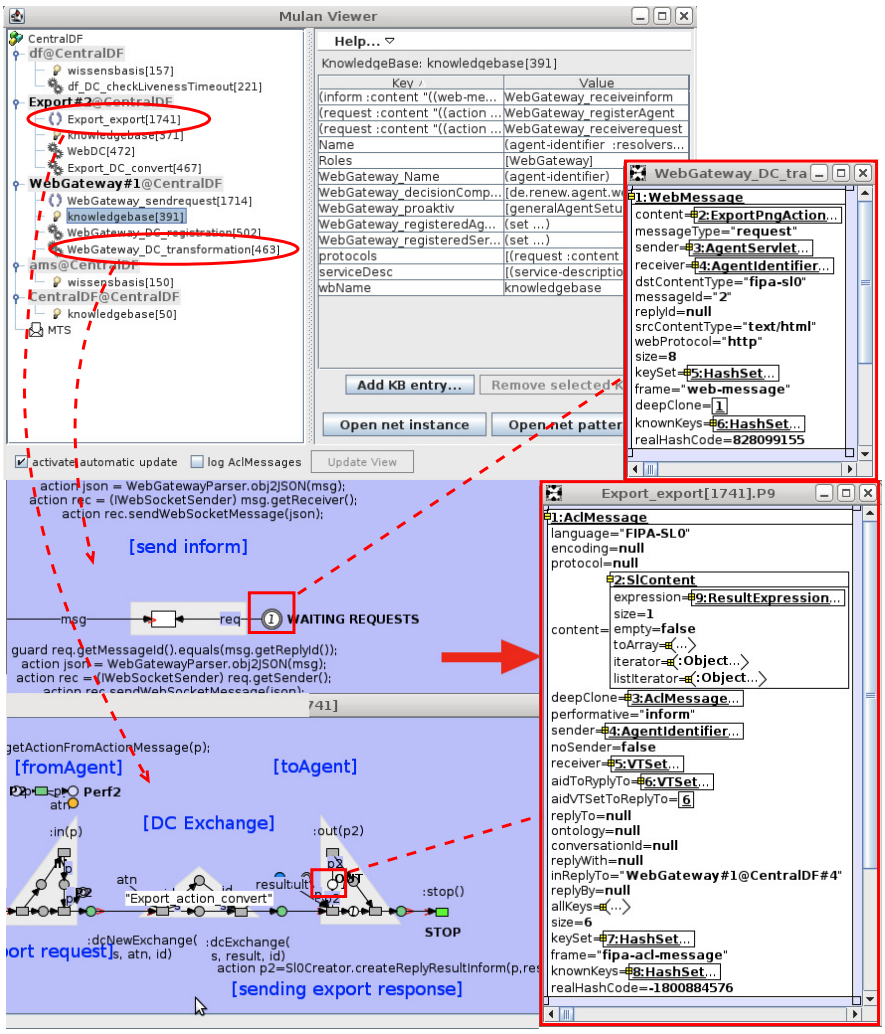

Figure 9. A screenshot of the agent application while running the export interaction.

The availability and the robustness of the Export and Diff Web services provided through a running instance of a Petri net application shows that our framework is already beyond a pure proof of concept.

5 Implementation

Although executable models tend to grow to a size that cannot be presented in all details, we present the implementation of the transformation component in

order to discuss the specification refinement that leads to the executable model. Often the process that leads incrementally to the executable model has been presented as *implementation through specification*. In this executable model we do not discuss the inscriptions and certain technical details such as the preparation and selection of the messages. Figure 10 shows an executable version of the WebGateway Transformation Component as an overview.[12] The details of the main parts are presented again in Figure 11. Compared to the abstract model shown in Figure 4 this model shows several refinements. First of all the interfaces – indicated by the dashed boxes – of the component have been duplicated. This results from the fact that our implementation allows for additional communication protocols and two connection types.

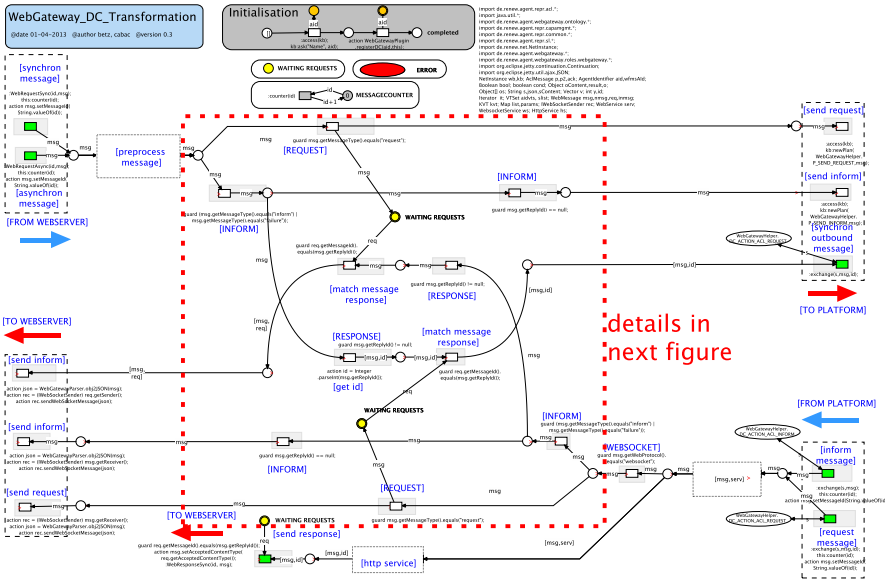

Figure 10. Implementation of the WebGateway Transformation Component.

In the abstract model we described the possibility to serve a typical request protocol. Thus, one party can send a request message and receive an answer to this in the form of an inform message. This protocol may be triggered from either side the web server interface or the agent system interface. Additionally, the implementation also allows for a simple inform message that has not been triggered and does not expect any follow-ups. Consequently, we have three out-

[12] The Petri nets is presented as a whole, in order to show the final result of the refinement process. Most of the details, such as declarations are only of minor importance. The main details are presented in a magnified version in Figure 11 for convenient inspection.

going transitions for these three different communication possibilities (request, inform as answer, simple inform) on each outgoing interface side. Additionally, we included the possibility to connect as a Web service in two possible ways. The first is the well-known http connection, the second, which is a websocket connection, allows for asynchronous communication through a bidirectional permanent connection link.

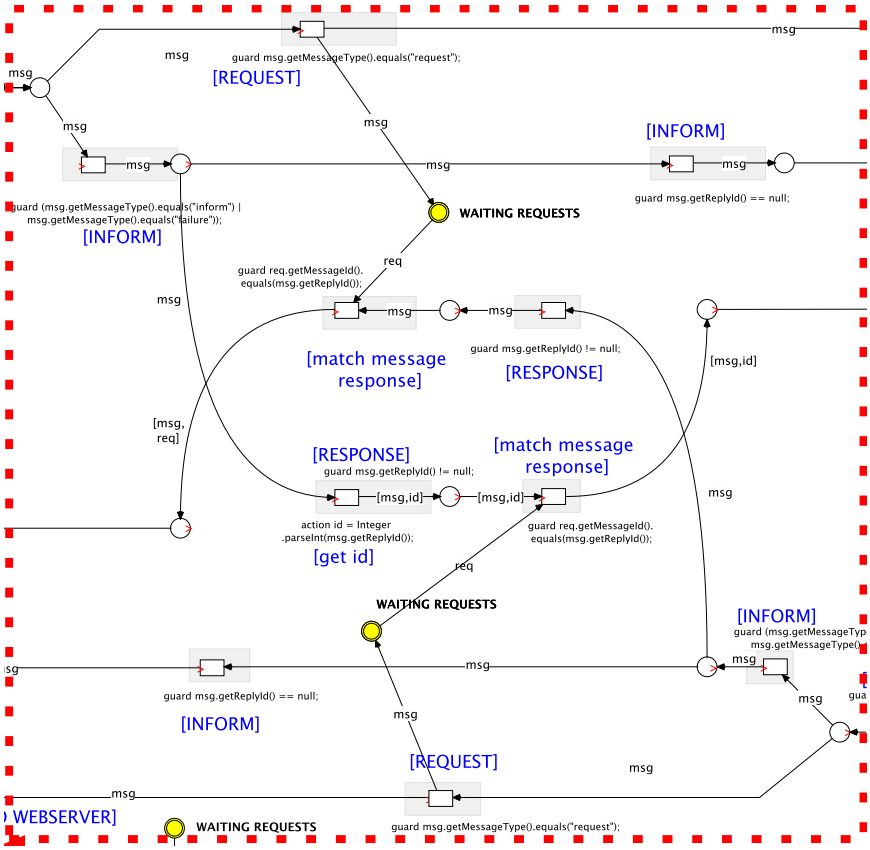

Figure 11. Fragment of the WebGateway Transformation Component.

In the center of the net – displayed in more detail in Figure 11 – one can prominently conceive the buffer places that hold the messages for matching answers to request and thus to determine the receiver. The buffer places are filled during the processing of the requests – constituting the first branch leading to the first of an outgoing interface. The answer collects the waiting message during processing – indicated by an arc – leading to the second interface transition. The last interface transition serves for the simple inform message. We have hidden

several parts of the original net – indicated by large dotted transitions. These transitions hide message processing and routing as well as the processing of the http answers.

6 Discussion

The presented example and the publicly available Web services and Web site demonstrate that the presented application is more than just a proof of concept. Especially the availability of the Export and Diff services as Web services has proven their usefulness in the straight-forward and seamless inclusion within several used instances of IPMEs (internal as well as public). We have also briefly mentioned the realization of Web-based GUIs where Web events related to the GUI components are transmitted between a browser and application agents (Agentlets). While this application of the WebGateway is still under development and still in an experimental stage, we have successfully made use of it in academic projects where students develop browser-based applications for collaboration support.

The choice of RESTful Web services in combination with WebSockets brings more flexibility to our gateway than strict WSU stack-based gateways can provide. Currently, we use rather simple service descriptions, which might hinder the automation of service workflows. A possible improvement in this area could be the integration of WADL[13], which plays a similar role for RESTful Web service as WSDL for SOAP Web services.

7 Related Work

Service oriented architectures especially in combination with Web applications is currently a popular field in the research community. Hence, the integration of Web services into multi-agent systems is a well researched topic with many interesting solutions. Such an approach that impacts on our introduced architectural design is presented by Greenwood et al. [14]. They introduce a *Web Service Integration Gateway Service* (WSGIS), which acts as a broker between the service participants and provides translation and routing mechanisms. Shafiq et al. [24] offers a slightly different solution that addresses the interconnection of FIPA-compliant multi-agent systems and Web services. Their approach rely on a middleware software that handles the communication of the service participants without any modification on the respective communication systems. In contrast to theses approaches, which are based on Web services that use the standard WSU[14] stack, the approach of Soto [19] makes use of the advantages of Web services that comply with the RESTful architecture [23]. He provides a *Web Service Message Transport Service* (WSMTS) for JADE platforms that is capable of handling FIPA-SL messages in XML representation. These messages

[13] Web Application Description Language (WADL): `http://java.net/projects/wadl`
[14] WSDL, SOAP, UDDI (WSU)

are extended with additional information that ensure an end-to-end communication with only one single message encoding. In this case, agents are able to register themselves with a specific address to publish their services as a REST service.

A still problematic issue, concerning the use and composition of RESTful services, is the change of state of a resource and the corresponding update of clients. Especially Web clients have to constantly send *GET* requests to check if the state has changed, which will result in heavy traffic and unnecessarily high load of Javascript. For a bidirectional communication, as used for instance in a User Interface Framework, Aghaee et al. [2, Section 5.2] recommend the use of W3C WebSockets [26]. The WebSocket API provides a mechanism to sent data in various representations (JSON, XML, etc.) to clients when the resource has changed.

An approach concerning the modeling of Web services using high-level Petri nets was given by Moldt et al. [21]. The authors introduce a four layer architecture that focuses on modeling the internal behavior of a Web service and provide a proposal for lifecycle management and interconnection of Web services.

With the approach presented in this paper we provide a prerequisite that enables us to verify the soundness of internal Web service processes by examination of agent interactions. In order to examine the composition of Web services we have to extend our approach to the external Web service interactions and their interfaces. A related approach is presented by Wolf [27] and his team at the University of Rostock. They provide formal models based on Petri nets to describe service interfaces and tools[15] that support the discovery and synthesis of matching service partners.

8 Conclusion and Future Works

In this paper, we present a gateway architecture that makes it possible to interconnect FIPA-compliant multi-agent systems and RESTful Web services. More specifically, it creates a bridge between Petri net-based (Mulan) agents and arbitrary Web service providers or clients. Its suitability for daily use has been proven by coupling image conversion and comparison services (provided by Mulan agents) with an integrated project management environment (running as a classic web server, using these services). Besides its useful functionality, the gateway as an artifact exemplifies the benefits of engineering Petri net-based software. The gateway architecture, its message routing core and its multi-agent interface are modeled and implemented in Java reference nets. We present the design of the core gateway functionality as coarse Petri net models, the integration of concrete functionality into these Petri nets – thus turning them into application code –, and the validation of certain application properties by using well-known Petri net analysis techniques.

On the practical side, the gateway broadens the range of applications for FIPA-compliant agents (and especially Mulan agents). Their functionality be-

[15] Service-Technology: `http://service-technology.org/tools/`

comes available for any Web service client, and they can refer to functionality provided by any other web service. The interaction with web services is restricted to the request-response pattern or just unidirectional information distribution, and thus not as feature-rich as the speech act-based communication in the multi-agent world. Nevertheless, these simple interaction patterns form the basis of any complex interaction and can thus be considered as sufficient for everyday use.

On the Petri net-based software engineering side, the tools and methods of the P*AOSE approach are evolving while we use them for the design and implementation of applications like the WebGateway. A major focus is currently put on validation and testing of the application's Petri net-based code artifacts.

The further use of our approach in future student projects and the continuous advancement of our agent-based collaboration platform will help to improve the gateway functionality and software engineering techniques step by step.

References

1. Gul Agha, Fiorella De Cindio, and Grzegorz Rozenberg, editors. *Advances in Petri Nets: Concurrent Object-Oriented Programming and Petri Nets*, volume 2001 of *Lecture Notes in Computer Science*. Springer-Verlag, 2001.
2. Saeed Aghaee and Cesare Pautasso. Mashup development with HTML5. In *Proceedings of the 3rd and 4th International Workshop on Web APIs and Services Mashups*, Mashups '09/'10, pages 10:1–10:8, New York, NY, USA, 2010. ACM.
3. Tobias Betz, Lawrence Cabac, and Matthias Güttler. Improving the development tool chain in the context of Petri net-based software development. In Michael Duvigneau, Daniel Moldt, and Kunihiko Hiraishi, editors, *Petri Nets and Software Engineering. International Workshop PNSE'11, Newcastle upon Tyne, UK, June 2011. Proceedings*, volume 723 of *CEUR Workshop Proceedings*, pages 167–178. CEUR-WS.org, June 2011.
4. Tobias Betz, Lawrence Cabac, and Matthias Wester-Ebbinghaus. Gateway architecture for Web-based agent services. In Franziska Klügl and Sascha Ossowski, editors, *Multiagent System Technologies*, volume 6973 of *Lecture Notes in Computer Science*, pages 165–172. Springer Berlin / Heidelberg, 2011.
5. Lawrence Cabac. *Modeling Petri Net-Based Multi-Agent Applications*, volume 5 of *Agent Technology – Theory and Applications*. Logos Verlag, Berlin, 2010.
6. Lawrence Cabac, Till Dörges, Michael Duvigneau, Daniel Moldt, Christine Reese, and Matthias Wester-Ebbinghaus. Agent models for concurrent software systems. In Ralph Bergmann and Gabriela Lindemann, editors, *Proceedings of the Sixth German Conference on Multiagent System Technologies, MATES'08*, volume 5244 of *Lecture Notes in Artificial Intelligence*, pages 37–48, Berlin Heidelberg New York, 2008. Springer-Verlag.
7. Lawrence Cabac, Michael Duvigneau, Daniel Moldt, and Heiko Rölke. Modeling dynamic architectures using nets-within-nets. In Gianfranco Ciardo and Philippe Darondeau, editors, *Applications and Theory of Petri Nets 2005. 26th International Conference, ICATPN 2005, Miami, USA, June 2005. Proceedings*, volume 3536 of *Lecture Notes in Computer Science*, pages 148–167, 2005.
8. Lawrence Cabac, Michael Duvigneau, Daniel Moldt, and Matthias Wester-Ebbinghaus. Towards unit testing for Java reference nets. In Robin Bergenthum and Jörg Desel, editors, *Algorithmen und Werkzeuge für Petrinetze. 18. Workshop AWPN 2011, Hagen, September 2011. Tagungsband*, pages 1–6, 2011.

9. Lawrence Cabac, Daniel Moldt, and Heiko Rölke. A proposal for structuring Petri net-based agent interaction protocols. In Wil van der Aalst and Eike Best, editors, *24th International Conference on Application and Theory of Petri Nets, Eindhoven, Netherlands, June 2003*, volume 2679 of *Lecture Notes in Computer Science*, pages 102–120. Springer-Verlag, June 2003.
10. Lawrence Cabac and Jan Schlüter. ImageNetDiff: A visual aid to support the discovery of differences in Petri nets. In *15. Workshop Algorithmen und Werkzeuge für Petrinetze, AWPN'08*, volume 380 of *CEUR Workshop Proceedings*, pages 93–98. Universität Rostock, September 2008.
11. Michael Duvigneau, Daniel Moldt, and Heiko Rölke. Concurrent architecture for a multi-agent platform. In Fausto Giunchiglia, James Odell, and Gerhard Weiß, editors, *Agent-Oriented Software Engineering. 3rd International Workshop, AOSE 2002, Bologna. Proceedings*, pages 147–159. ACM Press, July 2002.
12. Roy T. Fielding and Richard N. Taylor. Principled design of the modern Web architecture. *ACM Trans. Internet Technol.*, 2:115–150, May 2002.
13. OASIS (Organization for the Advancement of Structured Information Standards). BPEL: Web services business process execution language. Available at: http://bpel.xml.org/, 2012. Release 2.0: http://docs.oasis-open.org/wsbpel/2.0/OS/wsbpel-v2.0-OS.html.
14. D. Greenwood and M. Calisti. Engineering Web service - agent integration. In *Systems, Man and Cybernetics, 2004 IEEE International Conference on*, volume 2, pages 1918 – 1925 vol.2, october 2004.
15. Sebastian Hinz, Karsten Schmidt, and Christian Stahl. Transforming BPEL to Petri nets. *Lecture Notes in Computer Science*, 3649:220–235, 2005.
16. Ekkart Kindler, Vladimir Rubin, and Robert Wagner. Component tools: Integrating Petri nets with other formal methods. *Lecture Notes in Computer Science : Petri Nets and Other Models of Concurrency - ICATPN 2006, Volume 4024, 2006*, pages 37–56, 2006.
17. Michael Köhler, Daniel Moldt, and Heiko Rölke. Modelling the structure and behaviour of Petri net agents. In J.M. Colom and M. Koutny, editors, *Proceedings of the 22nd Conference on Application and Theory of Petri Nets 2001*, volume 2075 of *Lecture Notes in Computer Science*, pages 224–241. Springer-Verlag, 2001.
18. Olaf Kummer, Frank Wienberg, Michael Duvigneau, Jörn Schumacher, Michael Köhler, Daniel Moldt, Heiko Rölke, and Rüdiger Valk. An extensible editor and simulation engine for Petri nets: Renew. In Jordi Cortadella and Wolfgang Reisig, editors, *Applications and Theory of Petri Nets 2004. 25th International Conference, ICATPN 2004, Bologna, Italy, June 2004. Proceedings*, volume 3099 of *Lecture Notes in Computer Science*, pages 484–493, Berlin Heidelberg New York, June 2004. Springer.
19. Esteban León Soto. Agent communication using Web services, a new fipa message transport service for jade. In Paolo Petta, Jörg Müller, Matthias Klusch, and Michael Georgeff, editors, *Multiagent System Technologies*, volume 4687 of *Lecture Notes in Computer Science*, pages 73–84. Springer Berlin / Heidelberg, 2007. 10.1007/978-3-540-74949-3_7.
20. T. Miyamoto and S. Kumagai. An agent net approach to autonomous distributed systems. In *Proc. of 1996 IEEE Systems, Man, and Cybernetics, 14-17 October 1996, Beijing, China*, pages 3204–3209, October 1996.
21. Daniel Moldt, Sven Offermann, and Jan Ortmann. A Petri net-based architecture for web services. In Lawrence Cavedon, Ryszard Kowalczyk, Zakaria Maamar, David Martin, and Ingo Müller, editors, *Workshop on Service-Oriented Computing*

and Agent-Based Engineering, SOCABE 2005, Utrecht, Netherland, July 26, 2005. Proceedings, pages 33–40, 2005.
22. Julia Padberg and Hartmut Ehrig. Petri net modules in the transformation-based component framework. *Journal of Logic and Algebraic Programming, Vol 97 (1-2)*, pages 198–225, 2006.
23. Cesare Pautasso, Olaf Zimmermann, and Frank Leymann. Restful web services vs. "big" web services: making the right architectural decision. In *Proceeding of the 17th international conference on World Wide Web*, WWW '08, pages 805–814, New York, NY, USA, 2008. ACM.
24. M. Omair Shafiq, Ying Ding, and Dieter Fensel. Bridging multi agent systems and Web services: towards interoperability between software agents and semantic Web services. In *Enterprise Distributed Object Computing Conference, 2006. EDOC '06. 10th IEEE International*, pages 85 –96, october 2006.
25. Rüdiger Valk. Object Petri Nets – Using the Nets-within-Nets Paradigm. In Jörg Desel, Wolfgang Reisig, and Grzegorz Rozenberg, editors, *Advances in Petri Nets: Lectures on Concurrency and Petri Nets*, volume 3098 of *Lecture Notes in Computer Science*, pages 819–848. Springer-Verlag, Berlin Heidelberg New York, 2004.
26. World Wide Web Consortium (W3C). The websocket api editor's draft 6. Website, June 2011. http://dev.w3.org/html5/websockets.
27. Karsten Wolf. Does my service have partners? *LNCS ToPNoC*, 5460(II):152–171, March 2009. Special Issue on Concurrency in Process-Aware Information Systems.