DiaFlux: A Graphical Language for Computer-Interpretable Guidelines

Reinhard Hatko[1], Joachim Baumeister[2], Volker Belli[2], and Frank Puppe[1]

[1] Institute of Computer Science, University of Würzburg, Germany
{hatko, puppe}@informatik.uni-wuerzburg.de
[2] denkbares GmbH, Friedrich-Bergius-Ring 15, 97076 Würzburg, Germany
{joachim.baumeister, volker.belli}@denkbares.com

Abstract. In this paper, we introduce the formal representation language DiaFlux, that is simple and easy to use on the one hand. On the other hand, it allows for the definition of Computer-Interpretable Guidelines (CIGs), that can solve valuable tasks being executed in the clinical context. Further, we describe a wiki-driven development process using the stepwise formalization, and allowing for almost self-acquisition by the domain specialists. The applicability of the approach is demonstrated by a protocol for sepsis monitoring and treatment developed by a collaboration of clinicians.

1 Introduction

The work presented in this paper is conducted within the project “CliWE - Clinical Wiki Environments”[3]. We investigate languages, tools, and methodologies to collaboratively build Computer-Interpretable Guidelines (CIGs) by domain specialists themselves. The requirement concerning the language is the development of an explicit and executable representation of diagnostic knowledge for active decision-support systems. Furthermore, we create a development process for simple and effective knowledge acquisition by domain specialists. Ultimately, the final knowledge bases will be exported into mixed-initiative devices, that cooperate with the clinical user during the care process.

In recent years, knowledge engineering research has been heavily influenced by the emergence of Web 2.0 applications, such as wikis, blogs, and tagging systems. They provide a simplified access and a light-weight approach for knowledge acquisition. Furthermore, those systems usually allow for a distributed and (often) collaborative development process. One of the most popular examples is the wide-spread use of *wikis* as flexible knowledge management tools, both in personal life and business environments. Introducing the *semantic interpretation* of wikis, the development of Semantic Wikis [19] allows for a more formalized definition of the knowledge. Today, Semantic Wikis are mainly used for collaborative ontology development, by providing a flexible, web-based interface to build semantic applications [18].

[3] Funded by Drägerwerk AG & Co. KGaA, Lübeck, Germany, 2009-2012.

The main benefit of Semantic Wikis is their possibility to interweave different formalization types of knowledge in the same context. That way, ontological concept definitions are mixed with free text and images within the wiki articles. Such tacit knowledge often serves as documentation of the development process, or as pursuing additional information not representable in a more formal manner.

In this paper, we introduce the Semantic Wiki KnowWE, that was designed to build decision-support systems, and we propose the graphical language DiaFlux, for modeling of executable clinical protocols: The contributions of this language are its simple application for developing decision-support systems, since it only provides a limited number of intuitive language elements. Due to its simplicity it is possible to be used by domain specialists, and thus eases the application in the knowledge engineering process. Albeit its simplicity, a rich set of diagnostic elements can be integrated into the language, that are required to build sophisticated (medical) knowledge bases. Furthermore, the language allows for the incorporation of less explicit knowledge when needed. To allow for comfortable development of DiaFlux models, we introduce a visual editor integrated into the Semantic Wiki KnowWE.

The rest of the paper is organized as follows: Section 2 introduces the language DiaFlux for Computer-Interpretable Guidelines, using a simple example protocol. The reasoning engine for DiaFlux models is discussed in Section 3. Also the integration of the language into the Semantic Wiki KnowWE and the development process including stepwise formalization is described. Currently, the approach is evaluated by the development of a medical decision-support system. We describe the experiences of this case study in Section 4. The paper is summarized and concluded in Section 5, also giving an outlook for future work.

2 The DiaFlux Language

This section first describes the application scenario, then a short insight about guideline models in the diagnostics domain is given. Following, we introduce the representation language for clinical protocols, called *DiaFlux*.

2.1 Application Scenario

Clinical guidelines have shown their benefits by providing standardized treatment based on evidence-based medicine. Many textual guidelines are readily available and also shared through the internet, but rely on the proper application by the clinician during the actual care process. While clinical guidelines are mostly textual documents, clinical protocols are an implementation of them, offering a more specific procedure for diagnosis and treatment in a given clinical context [11]. Much effort has been put into the development of formal models for Computer-Interpretable Guidelines (CIGs) and protocols. Clinical decision-support systems, that execute CIGs, support the clinician in his decision-making at the point of care. A logical progression in guideline application is their automatic execution by medical devices, as for example described in [14].

Our application scenario are mixed-initiative devices that continuously monitor, diagnose and treat a patient in the setting of an Intensive Care Unit (ICU). Such semi-closed loop systems interact with the clinician during the care process. Both parties, the clinician and the device, are able to initiate actions on the patient. As some data is continuously available as a result of the monitoring task, continuous reasoning with this data is performed. An execution environment for automated clinical care in ICUs and the implementation of a guideline for weaning from mechanical ventilation are presented in [14].

Based on the described application scenario, we identified the following goals, that were persued during the development of the language DiaFlux:

1. *Repetitive execution of subtasks*: Monitoring involves the continuous observation of sensory data to detect fault states and initiate corrective action. Therefore, particular actions need to be performed in an iterative manner.
2. *Representation of time*: An integral part of the language is a built-in representation of time and temporal reasoning capabilities, e.g., to reason about the trajectory of sensory input.
3. *Truth maintenance*: To revise conclusions by the system and handle inputs by the user appropriately, a truth maintenance system (TMS) [7] is integrated into the execution engine. The TMS guarantees to select the appropriate actions based on the current state, especially in domains with high frequency data.
4. *Parallelism*: Subtasks with no fixed order and dependency can be allocated to additionally spawned threads of control, and thus allow for their parallel execution. Expressing parallelism is especially necessary for mixed-initiative systems, in which human and machine initiated actions are carried out concurrently.
5. *Modularity*: To alleviate the reuse of formalized knowledge, DiaFlux models are intended to be reused in different contexts. The modularization also helps to improve the maintainability of the knowledge base.
6. *Testability*: The evaluation of a knowledge base is an essential step prior to its productive use. We provide basic functionality for empirical testing and anomaly checks tailored to DiaFlux models.

2.2 Language Description

For the specification of a clinical protocol, two kinds of knowledge have to be effectively combined, namely declarative and procedural knowledge [5]. While the declarative part encompasses the facts and their relationships, the procedural one reflects the knowledge about how to perform a task, i.e., the correct sequence of actions. The declarative knowledge particularly consists of the terminology, i.e., findings, solutions, and sometimes also treatments and their interrelation. The procedural knowledge for diagnostics in a given domain is responsible for the decision which action to perform next, e.g., asking a question or carrying out a test. Each of these actions has a cost (monetary or associated risk) and a benefit (for establishing or excluding currently considered solutions) associated with it. Therefore, the choice of an appropriate sequence of actions is mandatory for efficient diagnosis and treatment. Guideline languages employ different kinds of Task Network Models to represent the procedural aspects [16]. They describe decisions, actions and constraints about their ordering in a guideline plan. Often, flowcharts are the underlying formalism to explicitly express control flow.

In DiaFlux models, the declarative knowledge is represented by a domain-specific ontology, which contains the definition of findings and solutions. This application ontology is an extension of the task ontology of diagnostic problem solving [1]. The ontology is generated using a special markup. Therefore, it is strongly formalized and provides the semantics necessary for executing the guideline. The procedural knowledge is represented by one or more flowcharts, consisting of nodes and edges. Nodes represent different kinds of actions. Edges connect nodes to create paths of possible actions. Edges can be guarded by conditions, that evaluate the state of the current session, and thus guide the course of the care process.

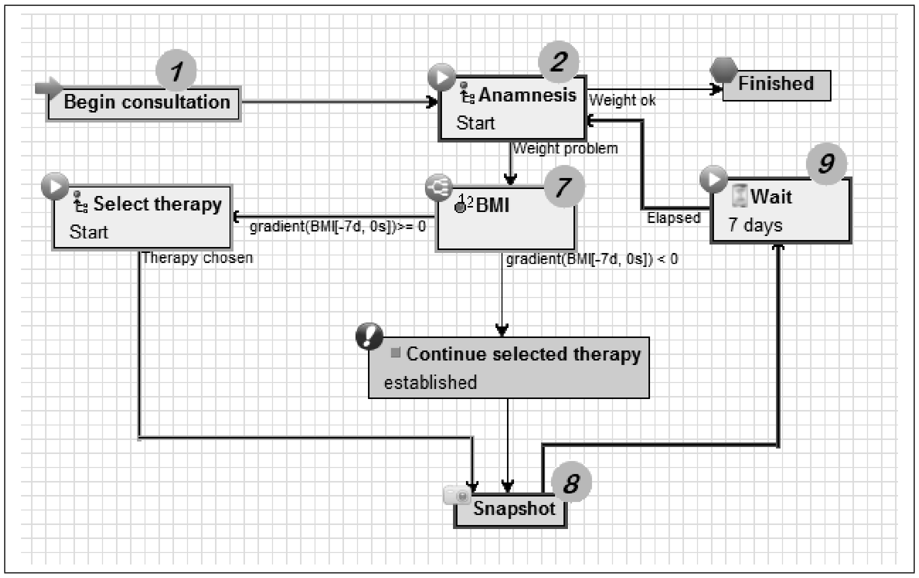

Fig. 1. The main model and starting point of a simple protocol for monitoring and treating overweight. The state of the current testing session is highlighted in green and yellow colors (black and grey in this image, respectively).

In the following, we informally describe the language elements, before we give an example using a simple protocol for the diagnosis and treatment of overweight, modeled in DiaFlux.

– **Start node:** A start node does not imply an action itself, but is a pseudo-node pointing to the node that represents the first action to take. Multiple start nodes can provide distinct entry points into a single DiaFlux model.
– **Test node:** Test nodes represent an action for carrying out a single test upon activation of the node at runtime. This may trigger a question, the user has to answer, or data to be automatically obtained by sensors or from a database. Furthermore, the collected information refines the knowledge about the patient state.
– **Solution node:** Solution nodes are used to set the rating of a solution based on the given inputs. Established solutions generate messages that are presented to the user and can, e.g., advice him to conduct some action.
– **Wait node:** Upon reaching a wait node, the execution of the protocol is suspended until the given period of time has elapsed.
– **Composed node:** DiaFlux models can be hierarchically structured, as already defined ones can be reused as modules, represented by a composed node. This fulfills the aforementioned goal of modularity.
– **Snapshot node:** Snapshot nodes can be used to mark distinct points of a model, at which the current execution state is saved, and truth maintenance will not influence the execution beyond this point. For further details see Section 3.
– **Abstraction node:** Abstraction nodes offer the possibility to create abstractions from available data, i.e., assigning values to abstraction questions. The results of abstraction nodes can be used to influence settings of the host device.
– **Exit node:** An exit node terminates the execution of a DiaFlux model, and returns the control flow to the superordinate model. To express different results of a model, several distinct labeled exit nodes are supported.
– **Comment node:** For the documentation of a protocol, comment nodes can be inserted at arbitrary positions. Though, they can be connected by edges, and be used to create semi-formal guidelines. They do not represent a specific action and are ignored during execution.

Figures 1 and 2 show parts of a protocol for the diagnosis and treatment of overweight modeled in DiaFlux. The main module, which is executed when a consultation session begins, is depicted in Figure 1. The execution of the module starts at the *start node* (1), labeled "Begin consultation", which is pointing to the *composed node* (2). On activation of this node, the submodule "Anamnesis" (cf. Figure 2) is called, and its start node labeled "Start" is activated. The execution of the current module is stalled, until the called submodule is processed. Reaching the *test node* "Height" (3), data is acquired from the user. After the value for the body height has been entered, the execution can progress to the test node "Weight". In contrast to the first test node, this one acquires new data each time it is activated, as the weight is supposed to change from one session to the next. Therefore, the specific testing action is "always ask" instead of "ask", as the first one triggers data acquisition even for inputs, that are already known, in order to update their value. After the value for "Weight" has been acquired, the

Abstraction node (4) calculates the body mass index (BMI), depending on the previously entered data, and assigns the value to the input "BMI". Depending on the value of the BMI the appropriate successor is chosen. For a value in the range of [18.5; 25[, the execution progresses to the *solution node* (5), which establishes the solution "Normal weight". The reached *exit node* (6), labeled "Weight ok", terminates the execution of the module, and returns the control flow to the superordinate protocol. For higher values of "BMI", the appriopriate solution is established, and the result "Weight problem" is returned as result of the "Anamnesis" module.

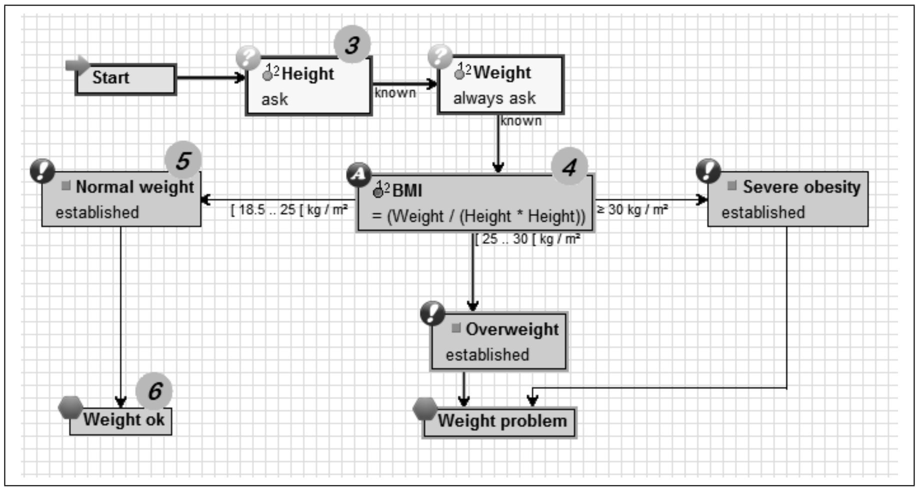

Fig. 2. The anamnesis submodel for acquiring data and establishing the current diagnosis.

After finishing the submodule, the appropriate successor of the composed node (2) is chosen based on the returned result. In case of the return value "Weight ok", the execution of the protocol ends by reaching the exit node "Finished", as there is no superordinate module. If a weight problem has been diagnosed, the treatment is chosen based on the history of values of the BMI. The *decision node* (7) tests for the gradient of BMI values. If the BMI is declining (i.e., the patient is loosing weight), the previously selected therapy is continued. Otherwise, another therapy is chosen within the module "Select Therapy"[4]. In both cases the *snapshot node* (8) is reached. On activation of this node, the execution state of the protocol is saved and truth maintenance will not retract any conclusion beyond this point. Furthermore, all active nodes on the incoming path are deactivated, to allow their repeated execution. A more thorough discus-

[4] A therapy is chosen during the first consultation, as the gradient of a single value is 0.

sion of this feature follows in Section 3. Next, the execution arrives at the *wait node* (9), which suspends the execution until the given time of 7 days has lapsed. Then, a second anamnesis is conducted, and the current BMI is calculated based on the newly acquired body weight value. If it has decreased, so will the BMI, and the current therapy is continued. Otherwise, a new therapy is selected, and applied until a normal body weight is obtained.

3 Reasoning with DiaFlux

The main focus of the DiaFlux language lies in the development of executable protocols. This section describes the according reasoning engine and the wiki-based development environment.

3.1 Architecture

The architecture of the DiaFlux execution engine comprises three main parts: First, a knowledge base, which contains the definition of declarative and procedural knowledge. Second, a blackboard represents the current state of the session. Last, a reasoning engine is responsible for executing the DiaFlux model. The protocol to be executed is defined within the knowledge base. It contains the application ontology and the flowcharts specifying the clinical care process. All findings describing the state of the current execution of the protocol are contained in the blackboard. These findings contain the input data as defined in the application ontology. A finding can either be entered by a user, or be derived by the system, or can contain data acquired by the system from connected sensors. To support temporal reasoning, all findings are time-stamped. The reasoning engine is responsible for interpreting the protocol depending on the current execution state, and for triggering its actions. Therefore, the reasoning engine is notified about all findings, that enter the blackboard.

3.2 Protocol Formalization

DiaFlux models are fully formalized and executable due to the underlying ontology of findings and solutions. Those are the basic elements of a diagnostics task, i.e., *findings* are used to derive particular *solutions*. A finding holds a value that is assigned to a specific input. An input consists of a name, a type (numerical or nominal), and a domain. Optionally, information like a unit of measurement and permitted value ranges can be specified. The defined inputs do not only contain *external* inputs (from sensors or a user) into the system, but also *internal* inputs, that describe the state of the execution. Internal inputs are derived by the system itself, and are then written to the blackboard, and fed back into the reasoning engine for further reasoning, e.g., abstractions calculated by according nodes. Findings of internal inputs can also be used to trigger external events, that have a measurable effect during the care process. For example, the current value of an internal input can be read by the appliance for adjusting certain operational parameters. Solutions are special output values, that are set by the system and presented to the user. They are used to, e.g., instruct the clinical user to accomplish a certain task.

3.3 Protocol Execution

The execution state of the protocol is given by the activation states of the nodes and edges contained in the flowcharts, as summarized in Table 1. Each node and edge can either be active or inactive. The state of a node is given by the state of its incoming edges. It is active, iff at least one of its incoming edges is active. An edge in turn is active, iff its starting node is active and the condition, it is guarded by, is evaluated to true. If an edge changes its state, the corresponding end node can also change its state. If an edge gets activated, its end node is activated, iff it was inactive before, otherwise nothing is done. Upon deactivation of an edge, its end point is checked for other active incoming edges supporting its state. If none exists, the node is deactivated. This in turn, can lead to further collapsing of formerly active paths. If a node is active, its associated action has been executed upon its activation as described in Section 2.2, e.g., acquisition of data or derivation of a solution. In case it gets deactived due to truth maintenance the action is undone again, which can, e.g., lead to findings being retracted from the blackboard.

Table 1. States and actions of DiaFlux model elements.

Model element	**Precondition for activation**	**On activation**	**On deactivation**
Node	At least one incoming edge is active	Action of the node is executed	Action of the node is retracted
Edge	1. Start node is active 2. Guard condition evaluates to true	Activate end node, iff it was not active before	Deactive end node, iff it has no other active incoming edge

The main idea of the DiaFlux reasoning engine is to distinguish between *volatile* and *non-volatile* actions. While non-volatile actions have an effect on the patient under treatment, volatile actions do not influence the care process. For example, deriving a finding which is not used to trigger external events can safely be undone. However, this is not possible, if a finding is used to alter the treatment of the patient. Therefore, volatile actions can safely be retracted by the truth maintenance system (TMS), without changing the course of the care process. This distinction makes it possible to react to changing patient states accordingly, as long as no action carried out has an effect on the patient.

The transition from volatile to non-volatile actions has to be marked by a special type of node, namely a *snapshot node*. Upon reaching a snapshot node during protocol execution, the contents of the blackboard (the contained findings) are protected from write access, i.e., they can no longer be retracted by the

TMS, but become the definite state for the further progression of the protocol execution. Additionally, all active nodes that are on the path to the activated snapshot node are set to inactive **without retracting** their action. They can then be activated again during the next monitoring cycle.

To select the entry point into the protocol one flowchart has to be marked as "autostart". This leads to the activation of all its start nodes, when the care process begins. From there on, the protocol execution is driven by findings entering the blackboard. Each of these findings together with its timestamp (for temporal reasoning and abstraction) is propagated to the reasoning engine. Then, all nodes and edges that use the associated input of a finding are checked for the validity of their state according to the new value of the finding.

3.4 The DiaFlux Modeling Environment

We created an implementation of the DiaFlux reasoning engine for the knowledge-based system d3web [3]. DiaFlux offers the possibility to model and execute protocols, that employ the declarative and inferential expressiveness provided by d3web. The development environment for DiaFlux models is integrated into the Semantic Wiki KnowWE (cf. Figure 3), using its plugin mechanism [17]. We created a graphical editor for easy modeling of the flowcharts. The editor is on the one hand able to reuse ontological concepts that are readily available in the wiki's knowledge base. Those can simply be dragged into the flowchart. Depending on the type of object (input, solution, DiaFlux model), a node of adequate type is created. On the other hand, the application ontology can be extended by creating new concepts from within the editor with a wizard. The model's source code is encoded in XML, and integrated into the corresponding wiki article, and saved and versioned together with it. This allows for further documentation of the protocol by tacit knowledge in the article. When the article is displayed in a web browser, the model visualization is rendered, instead of displaying its XML source code.

Modeling Process For the development of DiaFlux models, we propose the idea of the knowledge formalization continuum [2], where knowledge acquisition starts with informal knowledge, which is gradually refined until a formal representation is reached: At first, informal information can be collected in wiki articles, e.g. about goals of a protocol. During the next step, a first semi-formal flowchart can be created using only comment, start and exit nodes, and connecting edges (cf. Figure 4). At this stage of formalization, the flowcharts cannot be automatically executed, but "manually". For testing purposes the user can run through the flowchart by clicking on that outgoing edge of the active node, he wants to continue the pathway on. The taken pathway is highlighted for easier tracking. This is especially useful, when parallelism or hierarchically structured protocols are involved. The last step is the formalization into a DiaFlux model and the creation of the application ontology, resulting in a fully formalized and executable knowledge base. By following this process of gradual refinement, the entry barrier for domain specialists is quite low, while knowledge acquisition can start from the beginning.

Modeling Tools Collaborative development requires to track the changes of all participants. Therefore, a frequent task is to compare different versions of a protocol. For this purpose in general, wikis provide a textual diff comparing two versions of an article. As a diff of the XML source code is not very helpful for comparing a visual artifact like a flowchart, a more understandable diff is provided. On the one hand, a textual summary of the added, removed, and changed nodes and edges is generated. On the other hand, the previous and the current version of the DiaFlux model are shown next to each other, highlighting the changes in different colors for easy comparison, e.g. removed items are red in the previous version, added items are green in the current one, and changed items are highlighted in both versions.

After creating a knowledge base in KnowWE, it can directly be tested from the wiki article containing it, as d3web is also integrated into KnowWE. The current execution state of the protocol throughout the test session can be observed. The traversed pathway through the flowcharts is highlighted, in a similar manner as in the visual diff (cf. Figure 1). The currently active path is highlighted in green color. The path that was active, when the last snapshot has been taken, is marked with yellow color. This immediate visual feedback considerably eases the interactive testing of the knowledge base. Figures 1 and 2 show the state during the second consultation, after a period of 7 days. For the later stages of protocol development, complete test cases can be entered using a special type of table and then be executed.

3.5 Related Work

Many formal models for Computer-Interpretable Guidelines (CIGs) have been developed, an overview can be found in [16] and [12]. In the variety of CIG models, each has its own focus, e.g., GLIF [4] concentrates on the sharability of guidelines between various institutions, while PRO*forma* [8] focuses on assisting patient care through active decision support [5]. The DiaFlux language emphasizes compactness, offering a limited number of intuitive language elements, enabling knowledge acquisition by domain specialists themselves.

A related wiki environment for the collaborative creation of clinical guidelines is the Modelling Wiki (MoKi) [10], based on Semantic Media Wiki [13]. Originally, it was designed for the creation of enterprise models using a visual editor, but it also has been used in the Oncocure project [6] to acquire clinical protocols for breast cancer treatment. Therefore, templates were defined within the wiki and later their content was exported into skeletal Asbru plans [15]. Though MoKi's visual editing capabilities for business processes, they were not employed to graphically model guidelines. Furthermore, the created Asbru plans are currently not executable within the wiki.

4 Case Study

4.1 The Sepsis Protocol

In the context of the project "CliWE", we used a prototype of the clinical wiki environment for the development of a protocol covering the monitoring and therapy of sepsis. Sepsis is a syndrome of a systemic inflammation of the whole body with a high mortality (30 to 60%). There are two main problems in sepsis therapy. First, it is essential to detect, that a patient fulfills sepsis criteria and second, if sepsis is diagnosed a complex medical therapy has to be initiated quickly. Today, so called patient data management systems are available in many intensive care units. With these systems, medical data are electronically available. In this context a clinical decision support system may be a reasonable solution for the above outlined practical problems, monitoring all patients for sepsis and support the physician in the initiation of sepsis treatment.

The knowledge base was developed in accordance to the official guideline of the German Sepsis Society [9]. It is a textual guideline of about 80 pages describing the prevention, diagnosis, and therapy of sepsis. Our formalization of the guideline contains so far the diagnostics and parts of the therapy together with some common tasks for patient admission (cf. Figure 3). At the moment it contains about 50 nodes in eight modules with several possible pathways, depending on how the diagnosis can exactly be established and the course of the therapy. The upper part of the main DiaFlux model contains knowledge about the decision making and the lower part contains knowledge about the treatment.

The diagnosis task involves the assessment of up to eight clinical parameters (conducted in the modules "Septic parameters" and "Extended septic parameters") and an established or suspected infection. The monitoring is repeated until a sepsis can be established within different cycles, depending on which parameters are acquired and their evaluation. If there is sufficient evidence to support a suspected sepsis, then a warning to the clinician is generated. If the clinician agrees with the conclusion, the diagnosis "Sepsis" is established and instructions for starting the therapy are given. The treatment for sepsis consists of the three bundles *causal therapy* (treating the cause of the infection), *supportive therapy* (stabilizing the patients circulation) and *adjunctive therapy* (supporting fighting off the infection). Those bundles are modeled as self-contained modules and reused as composed nodes in the main module.

4.2 Experiences

The knowledge acquisition mainly took place in two workshops, approximately six hours each, involving two domain experts. The DiaFlux editor was handled by a knowledge engineer, entering the knowledge artifacts provided by the domain specialists. The remaining participants followed the authoring process on a projector.

During the first session, we followed the idea of the knowledge formalization continuum and started with textual descriptions of most modules. As the second

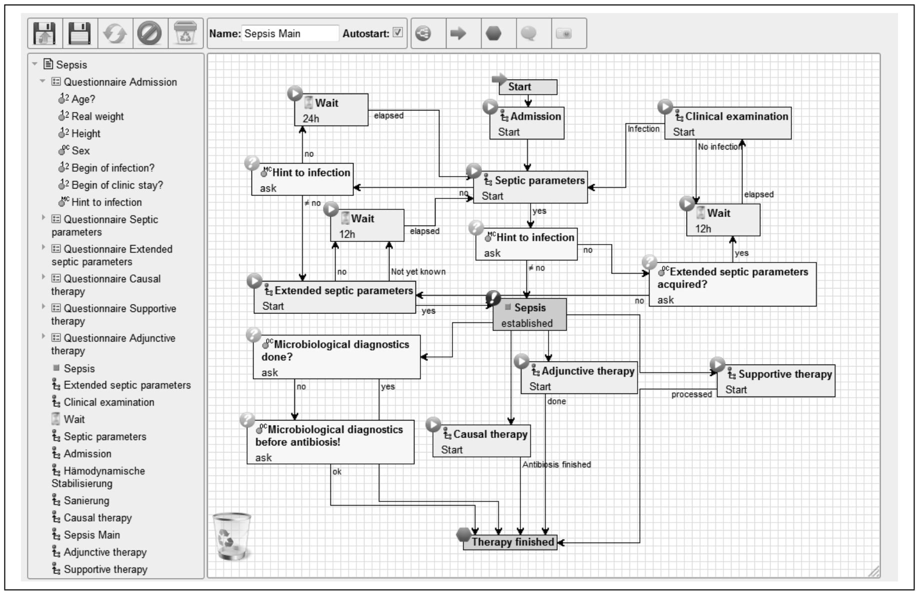

Fig. 3. The main module of the sepsis monitoring and treatment protocol, opened in the web-based editor. On the left side the declarative knowledge, that has already been defined, is contained and can be used to model the process.

step, we created semi-formal flowcharts giving an outline of the protocol, as exemplified in Figure 4. Next, we started to further formalize these flowcharts into executable DiaFlux models and to create the according declarative knowledge. The second session began with the acquisition of test cases of typical sepsis patients. As they were only informally entered in a wiki article and not executable so far, we stepped manually through the model by highlighting the correct pathway. The found inconsistencies were corrected during the second half of the session, together with further elaboration of the knowledge base. In a third session of about one hour, one of the experts created a small module by himself, while being observed by a knowledge engineer. The expert shared his screen using an internet screen sharing software, and was supported in formalizing the knowledge and the usage of the DiaFlux editor.

Overall, the wiki-based approach showed its applicability and usefulness, as the combination of formal and informal knowledge and its gradual refinement was intensely used during the acquisition of the protocol and the test cases. Further, the developed knowledge base was accessible to all participants immediately after the workshops, as it took place in a password protected wiki, which can be accessed over the internet.

So far, the knowledge acquisition was conducted in workshops involving domain experts and knowledge engineers. After the initial workshops and the suc-

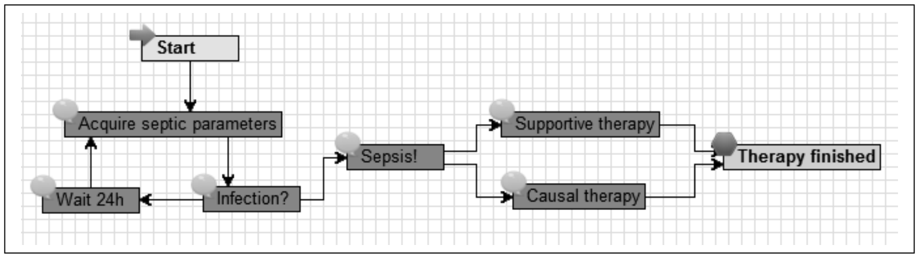

Fig. 4. An early semi-formal version of the sepsis protocol.

cessful tele-knowledge acquisition session, we are confident to proceed with further workshops, that require less support by the knowledge engineers.

5 Conclusion

This paper presented work in the context of the project "CliWE - Clinical Wiki Environments" for collaborative development and evolution of clinical decision-support systems. We introduced the language DiaFlux that can incorporate declarative and procedural diagnostic knowledge for modeling executable clinical protocols. Its main focus is the construction of protocols that are executed by mixed-initiative appliances in the setting of ICUs. The development environment is integrated into the Semantic Wiki KnowWE to support the collaborative development by a community of experts. The case study demonstrated the applicability and benefits of the approach during the development of a clinical protocol for sepsis monitoring and treatment. Due to the wiki-based approach, the knowledge can evolve easily. It is accessible without depending on specialized software, as long as an internet connection is available. Furthermore, domain specialists can almost instantly start contributing. Formalization of the knowledge can then happen at a later time, after familiarizing with the semantics.

As next steps we plan the integration of refactoring capabilities into the editor, for the easier evolution of DiaFlux models. We will also enhance the tool support for the gradual knowledge formalization.

References

1. Baumeister, J., Reutelshoefer, J., Puppe, F.: KnowWE: A semantic wiki for knowledge engineering. Applied Intelligence (2010)
2. Baumeister, J., Reutelshoefer, J., Puppe, F.: Engineering intelligent systems on the knowledge formalization continuum. International Journal of Applied Mathematics and Computer Science (AMCS) 21(1) (2011)
3. Baumeister, J., et al.: The knowledge modeling environment d3web.KnowME. open-source at: http://d3web.sourceforge.net (2008)

4. Boxwala, A.A., Peleg, M., Tu, S., Ogunyemi, O., Zeng, Q.T., Wang, D., Patel, V.L., Greenes, R.A., Shortliffe, E.H.: GLIF3: a representation format for sharable computer-interpretable clinical practice guidelines. J. of Biomedical Informatics 37(3), 147–161 (2004)
5. de Clercq, P., Kaiser, K., Hasman, A.: Computer-interpretable guideline formalisms. In: ten Teije, A., Miksch, S., Lucas, P. (eds.) Computer-based Medical Guidelines and Protocols: A Primer and Current Trends, pp. 22–43. IOS Press, Amsterdam, The Netherlands (2008)
6. Eccher, C., Rospocher, M., Seyfang, A., Ferro, A., Miksch, S.: Modeling clinical protocols using Semantic MediaWiki: the case of the oncocure project. In: K4HelP: ECAI 2008 Workshop on the Knowledge Management for Healthcare Processes. pp. 20–24. University of Patras (2008)
7. Forbus, K.D., de Kleer, J.: Building problem solvers. MIT Press, Cambridge, MA, USA (1993)
8. Fox, J., Johns, N., Rahmanzadeh, A.: Disseminating medical knowledge: the proforma approach. Artificial Intelligence in Medicine 14(1-2), 157 – 182 (1998), selected Papers from AIME '97
9. German Sepsis-Society: Sepsis guideline. http://www.sepsis-gesellschaft.de/DSG/Englisch
10. Ghidini, C., Kump, B., Lindstaedt, S.N., Mahbub, N., Pammer, V., Rospocher, M., Serafini, L.: MoKi: The enterprise modelling wiki. In: ESWC'09: The Semantic Web: Research and Applications. LNCS, vol. 5554, pp. 831–835. Springer (2009)
11. Hommersom, A., Groot, P., Lucas, P., Marcos, M., Martínez-Salvador, B.: A constraint-based approach to medical guidelines and protocols. In: Teije, A.t., Miksch, S., Lucas, P. (eds.) Computer-based Medical Guidelines and Protocols: A Primer and Current Trends, Studies in Health Technology and Informatics, vol. 139, pp. 213–222. IOS Press (2008)
12. Isern, D., Moreno, A.: Computer-based execution of clinical guidelines: A review. International Journal of Medical Informatics 77(12), 787 – 808 (2008)
13. Krötzsch, M., Vrandecić, D., Völkel, M.: Semantic MediaWiki. In: ISWC'06: Proceedings of the 5th International Semantic Web Conference, LNAI 4273. pp. 935–942. Springer, Berlin (2006)
14. Mersmann, S., Dojat, M.: SmartCare^{tm} - automated clinical guidelines in critical care. In: ECAI'04/PAIS'04: Proceedings of the 16th European Conference on Artificial Intelligence, including Prestigious Applications of Intelligent Systems. pp. 745–749. IOS Press, Valencia, Spain (2004)
15. Miksch, S., Shahar, Y., Johnson, P.: Asbru: A task-specific, intention-based, and time-oriented language for representing skeletal plans. In: UK, Open University. pp. 9–1 (1997)
16. Peleg, M., Tu, S., Bury, J., Ciccarese, P., Fox, J., Greenes, R.A., Miksch, S., Quaglini, S., Seyfang, A., Shortliffe, E.H., Stefanelli, M., et al.: Comparing computer-interpretable guideline models: A case-study approach. JAMIA 10 (2003)
17. Reutelshoefer, J., Lemmerich, F., Haupt, F., Baumeister, J.: An extensible semantic wiki architecture. In: SemWiki'09: Fourth Workshop on Semantic Wikis – The Semantic Wiki Web (CEUR proceedings 464) (2009)
18. Schaffert, S.: IkeWiki: A semantic wiki for collaborative knowledge management. In: STICA'06: 1st International Workshop on Semantic Technologies in Collaborative Applications. Manchester, UK (2006)
19. Schaffert, S., Bry, F., Baumeister, J., Kiesel, M.: Semantic wikis. IEEE Software 25(4), 8–11 (2008)