A Problem Representation Approach for Decision Support Systems

Sean C. G. Kern
Air Force Institute of Technology
Department of Electrical and Computer Engineering
Wright-Patterson AFB OH 45433
sean.kern@afit.af.mil

Michael T. Cox
Wright State University
Department of Computer Science and Engineering
Dayton, OH 45435-0001
mcox@cs.wright.edu

Abstract

The choice to include the human in the decision process affects four key areas of system design: problem representation, system analysis and design, solution technique selection, and interface requirements specification. We introduce a design methodology that captures the necessary choices associated with each of these areas. In particular we show how this methodology is applied to the problem representation process for an actual decision support system for satellite operations scheduling. We highlight the main issues related to problem representation in the large and how these issues seek to capitalize on the strengths of the computer system and the human user. We discuss our implementation of a scheduling problem representation scheme that embodies these concerns and we show the decision support advantages gained as a result of this representation.

Introduction

The goal of *mixed-initiative systems* is to synergistically combine the capabilities of the human user and the computer system with the intent to produce higher quality solutions than either the human or the computer can produce independently (Cox & Veloso, 1997; Ferguson, Allen & Miller, 1996; Oates & Cohen, 1994). In one form mixed-initiative systems act as decision support aids, assisting the user in reaching the most optimal problem solution.

Supporting the user's ability to monitor the actions of the system and to guide the decision process of the system are two key considerations in the successful design of a decision support system. Both of these points rely on the correct specification of human-computer interaction points. Traditional, computer-centered system design approaches do not do this well, if at all, and are insufficient for the design of decision support systems. These approaches typically leave the definition of human-computer interaction points till after the component and system level designs are complete. This is too late however since the component and system level design decisions can impose inflexible constraints on the choice of the human-computer interaction points. This often leads to the design of human-computer interaction points that are only "good enough." These approaches result in ill-conceived problem representations and poor user-system interaction points because the system lacks the underlying architecture to support these constructs efficiently. Decision support systems require a new, human-centered design approach rather than the traditional computer-centered approaches.

Human-centered design refers to the process of designing a software system that enables the human to monitor the actions of the system and provide varying degrees of guidance regarding the decisions made by the system. The ability of the user to monitor system actions is critical to the user's understanding of the current problem space. Supplying the user with knowledge of the problem space is essential for providing assistance to the user as the context of the user's interaction approaches the manual end of the *guidance continuum*. The guidance continuum may range from full automation, to explicit manual operation, to some combination of these.

The methodology shown in Figure 1 provides an alternative to the computer-centered design techniques. It is a human-centered design process that systematically decomposes the design problem into an ordered sequence of design decisions that are sensitive to the level of user involvement in the problem solution process (Kern, 2000). The result of each design decision acts as input to the subsequent design decision.

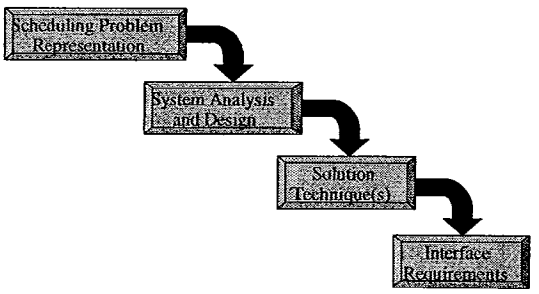

Figure 1: Scheduling system design methodology

The design decisions surrounding the scheduling problem representation step identify and classify the object classes of interest in the problem domain. It further organizes these classes based on the responsibilities of the

participants in the solution process. A participant, for example, may be responsible for describing the problem state while another may act on the problem state to achieve some system-wide goal. The result is a *process level description* that clarifies the relationships between process participants with regard to the objects in the problem domain. This allows for the case that more than one human or computer may be responsible for the overall problem specification and solution.

The system analysis and design step uses this process level description to select an appropriate architecture and system design methodology. Based on the chosen architecture and the levels of user guidance and monitoring expected, the designer can choose the appropriate class of problem solution techniques. The culmination of the previous steps results in an interface requirements definition. The interface requirements definition specifies the manner in which the user can monitor and guide the actions of the decision support system. These design steps must be defined in a manner that supports the user's problem understanding, interaction, and ability to respond to the problem environment.

In our approach we represent the problem in a way that provides intuition to the user, uses common terms, and ensures understanding. We also state the case for choosing a mixed-initiative, agent-oriented design approach. This choice influences the underlying system architecture and ensures mechanisms to support the appropriate levels of user monitoring and guidance exists. User interfaces are mapped directly to the architectural components, guaranteeing the emergence of a tightly coupled human/agent team. This tight coupling is at the heart of a well-designed, distributed decision support process and is central to the methodology this research presents.

The focus for this discussion is the problem representation step. Section 2 discusses the concept of a scheduling problem representation. We highlight the main issues related to problem representation and how these issues seek to capitalize on the strengths of the computer system and the human user. Section 3 describes our implementation of the scheduling problem representation scheme in our prototype application and discusses the decision support advantages gained as a result of this representation. Finally, section 4 concludes with a brief discussion and future research.

Problem Representation

Several object-oriented frameworks exist that aid in the development of scheduling systems (Smith & Becker, 1997; Beck, 1998; Cesta, Oddi, & Susi, 1999; Dorn, Girsch, & Vidakis, 1999). They take advantage of the large body of theoretical work in scheduling to identify abstract objects common to scheduling problems in the large. Smith and Becker represent these abstract objects in their Abstract Scheduling Domain Model depicted in Figure 2 (Smith & Becker, 1997).

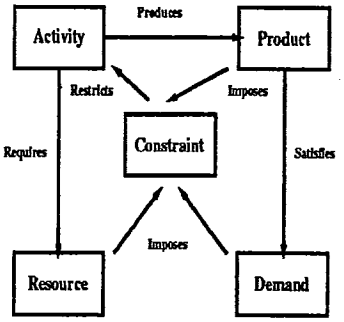

Figure 2: Abstract Scheduling-Domain Representation

Our research extends this abstract domain model and provides a methodology for using this abstract domain model to assist in the problem representation process. The following subsections describe the process we used to represent the satellite operations scheduling problem in our prototype application and the extensions we made to the Abstract Scheduling Domain Model.

The Problem Representation Process

Current literature does well to describe the types of objects that comprise the abstract scheduling problem. However there is no documented process for representing the scheduling problem of a given domain with respect to the abstract scheduling problem. We present a process here that capitalizes on the existing theoretic scheduling research to aid in the problem representation process for a given domain.

The schedule problem representation process presented here requires the Abstract Schedule Domain Model and domain expertise as input. The Abstract Scheduling Domain Model is used as the representation of choice because it clearly depicts the abstract objects and their relations. In addition other frameworks surveyed implicitly support similar forms of this model. Expert interviews, analysis of the existing domain scheduling process, and other methods of gathering domain expertise are good sources of information for constructing the scheduling problem representation.

The first phase in problem representation is a domain analysis of the problem domain. Domain analysis results in the identification of objects central to the scheduling problem. The second phase uses the abstract objects (e.g., ACTIVITY, RESOURCE) in the Abstract Scheduling Domain Model as *classification categories* to group the objects. The result of this classification is a specification of the concrete scheduling objects that represent completely

and accurately the given scheduling problem. The third phase organizes these classified concrete objects with regard to the participants in the system. Three categories are defined to represent participant responsibilities in the system. We identified the participants responsible for introducing the objects into the system and those participants responsible for processing and managing the objects in the system. The final phase introduces the flow of schedule objects between participants. The result is a process level description of the given scheduling problem. The overall output of the problem representation step is a complete representation of the scheduling domain at the concrete object level and at the process level. Figure 3 shows the phases in this step.

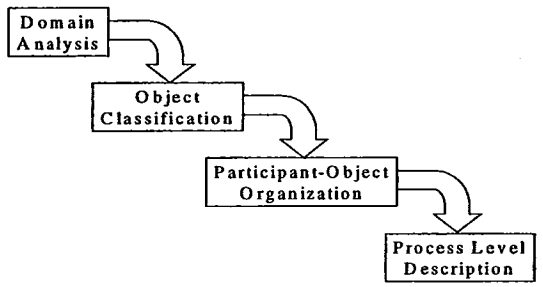

Figure 3: Scheduling-Problem Representation

The following steps summarize the problem representation approach taken in our research.

1. Identify inputs and outputs of the current process (e.g., schedule request)
2. Select a corresponding abstract object from the Abstract Scheduling Domain Model
3. Compare inputs and outputs to an abstract object's properties and capabilities and find matches (e.g., schedule request = DEMAND)
4. Follow a relation link (e.g., ACTIVITY produces PRODUCT) from the abstract object selected in Step 2 and repeat Step 3 until all relevant concrete objects are identified and classified.
5. Organize concrete objects by process participant (e.g., Participant A is responsible for all resources)

The strengths of this approach are threefold. First, it uses common software engineering techniques (i.e., domain analysis and object classification) that are familiar to software developers in general, aiding the overall system design process. Second, it incorporates a concrete scheduling object organization scheme as well as a process level organization scheme that aids in the selection of appropriate component level and system architectures in the following design methodology step. Third, it represents the concrete scheduling objects in a manner that is intuitively clear to the user. This representation process also educates the user in a manner that enables the user to understand the underlying actions of the system thereby engendering a sense of trust in the system's actions. Understanding the problem space and trust are two key benefits of this process and are considered desirable traits of mixed-initiative systems (Burstein & McDermott, 1994; Brown & Cox, 1999).

Extensions to the Abstract Scheduling Domain Model

The existing Abstract Scheduling Domain Model proved valuable except in two cases. One of the underlying goals of our research is to assist the user in scheduling activities by automatically generating activities using available system data. The properties and capabilities defined for the abstract DEMAND scheduling object in the Abstract Scheduling Domain Model does not include a provision for modeling DEMANDS as automatic entities. In fact none of the systems surveyed addressed the issue of automatic demand generation. We therefore include a property for a DEMAND that enables the system to determine whether or not a DEMAND should be automatically generated for subsequent processing. When a demand is defined in the system as automatic, it is further defined in domain-dependent terms based on the frequency and/or conditions that govern its automatic generation. These domain dependencies are not considered extensions to the model, but must be considered nonetheless by the system designer. This is discussed further in Section 3.

The other case where it was necessary to extend the Abstract Scheduling Domain Model is with respect to the properties and capabilities of a RESOURCE. To assist the user in generating necessary schedule demands, it is helpful for a resource to have a "sense of self". By actively monitoring conditions in the system, the resource can "ask" the system to generate a demand on its behalf. Conversely the system can poll the resource for any pending demands and the resource can respond based on its internal state. In this way the resource acts as an active agent in the system. This is not inconsistent with the abstract RESOURCE object. We do not suggest a core of abstract capabilities or properties to include in the model to represent active and passive resources. We do suggest that a provision be made that recognizes the decision support benefits of this approach.

Implementation

Following the approach shown in Figure 1 we implemented a prototype scheduling system called SOSA, Satellite Operations Scheduling Assistant. It is an object-oriented class library written in the Java programming language consisting of five Java packages comprising the software agents, messaging mechanism and conversation protocols, concrete and abstract scheduling objects, and interface components of the system. We identified, classified, and organized the concrete scheduling objects in the satellite operations domain in the manner described in Section 2. The result is a problem representation that enables the system to provide decision support services to the user as enabling the user to monitor and guide the system services. The following example supports these statements.

In our example domain, the user currently submits a request to the human scheduler to schedule an activity. Scheduling theory does not support the notion of an activity as the unit of submission; a demand is the appropriate way to represent a user requirement. The user should instead submit a demand to the system. The system in turn processes the demand resulting in one or more products that satisfy the demand. The products are further processed to generate one or more atomic activities that, when executed, produce the needed products. Using our problem representation approach, the user moves away from an *activity-centric* scheduling mentality to a *demand-centric* one. The user is no longer specifying the individual activities he needs to achieve a system goal state, but instead submits a demand that automatically equates to the activities that move the system to the desired goal state.

There are several benefits to this approach. First, activity-centric scheduling requires the user to know the individual activities necessary for achieving the goal system state. If activity-types are added or deleted the user needs to be aware of this each time he attempts to schedule activities to achieve a desired system state. In demand-centric scheduling these facts are abstracted from the user and handled by the system. Second, in activity-centric scheduling the user is required to know the temporal relations between these activities (if any exist). If the requirement for Activity B to follow Activity A was 4 hours and is now changed to 3 hours, the user needs to be aware of this for all future submissions of Activities A and B. In the demand-centric approach the system again abstracts this knowledge from the user. Third, the user typically needs to know the resources required by each activity in order to submit an activity request. In demand-centric scheduling the system tracks the resources required for each activity. The problems with activity-centric scheduling are further exacerbated when dealing with novice users.

Our prototype system modeled automatic demands as *location-specific*, *time-specific*, or a combination of the two. Location-specific demands are automatically generated by the system whenever the physical location of the satellite warrants. A satellite's attitude in relation to the sun and/or moon is one type of location-specific consideration. Time-specific demands are generated on a regularly recurring basis (i.e., the first Monday of the month). Finally a combination demand is automatically generated for a specific time as long as the satellite's location is/is not within given location thresholds.

The move towards viewing resources as active objects and relying on the resources to generate demands on their behalf is a direct result of the choice to model demands as automatic or manual. We recognized early that there were state properties associated with the satellite resources that could aid in our goal to provide automated decision support services to the user. This is particularly true since the satellite in our domain is the "center of attention" so to speak. The satellites are "aware" of their physical locations and the current system time. When conditions for an automatic demand are met, the satellite submits a demand to the system. This is a significant decision support service since the human satellite engineer is relieved from manually verifying the conditions for each satellite in the constellation.

The focus of the schedule representation problem now shifts from representing demands to representing products in the system. The products are called *mission types* in the satellite operations domain and are composed of activities, called *command plans*. Additionally mission types contain the temporal relations that exist between the command plans. We developed a user interface that allows the user to add, remove, and modify mission types in the system. This is equivalent to the user adding, removing, and modifying a portion of the system rules used for scheduling. This is the abstraction mechanism we employed to remove the common user from memorizing the activities and temporal relations required to achieve a desired goal state. The expert user can specify the appropriate scheduling rules and all subsequent users, novice or otherwise, can benefit. Figure 4 shows the Mission Type Editor interface.

In the example depicted in Figure 4, the *ranging mission type* is comprised of six *ranging command plans*. Each ranging command plan starts four hours after the previous one. The user has the option to add command plans from the Command Plan Library, deleting command plans from the ranging mission type, or modify the temporal relations between the respective command plans. The main point to make here is that once a mission type is defined, the user no longer is required to know the number and type of command plans required to complete a mission or their respective temporal relations. This approach also provides an intuitive way to describe the products that are produced by the system and the activities that produce those products.

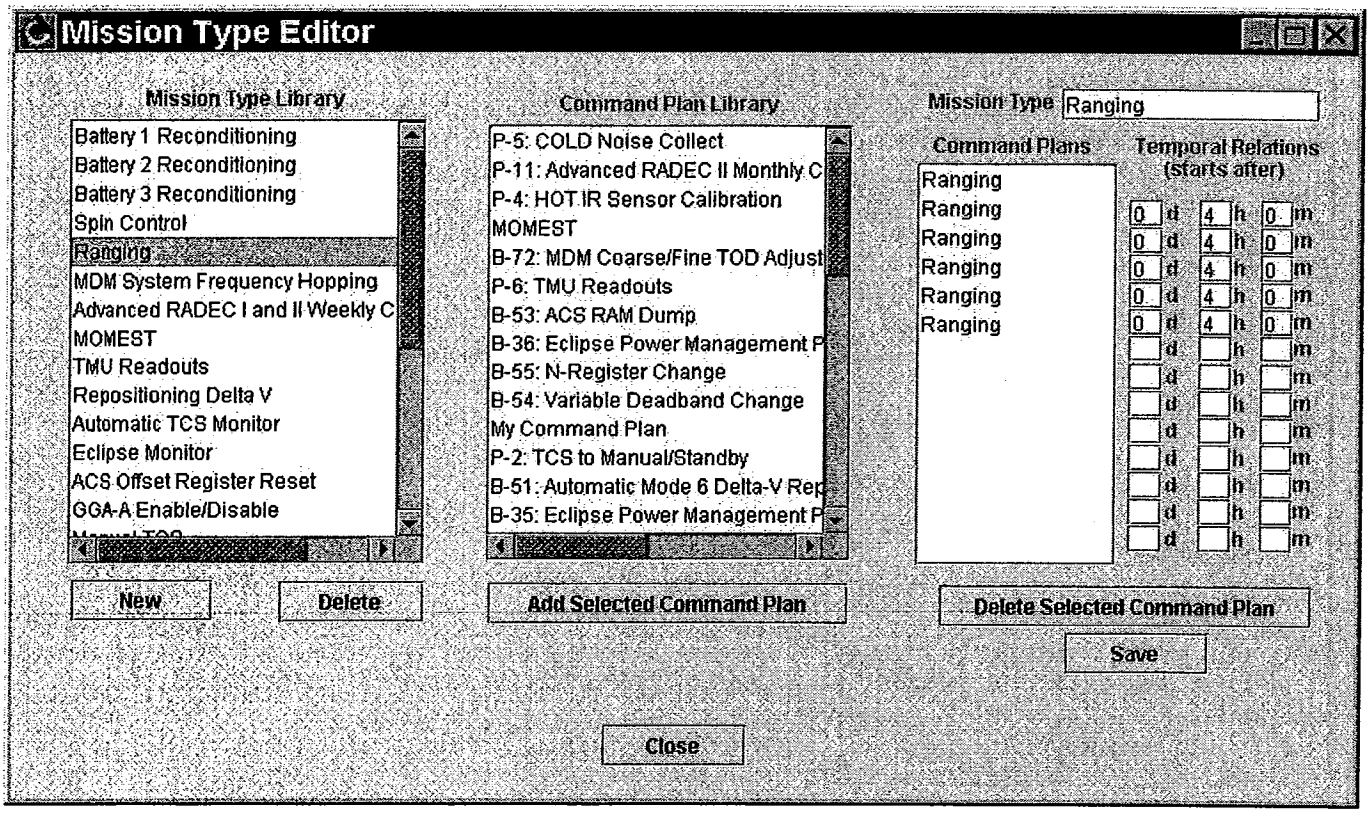

Figure 4: The Mission Type Editor

Conclusions

We introduced a software design process that can be used to design a decision support system for scheduling. We emphasized the necessity for identifying desired levels of user monitoring and guidance of the decision support system early in the design process. We showed the major steps in the design process and examined in detail the first of these steps (i.e., problem representation). We described a process for representing the scheduling problem that identifies, classifies, and organizes concrete scheduling objects. This leads to a process level description that is used in the subsequent System Analysis and Design step. We took advantage of existing work in scheduling theory using the Abstract Schedule Domain Model and explained our need to extend the model to account for our goal of automatic demand generation. This was accomplished by adding an automatic property to the DEMAND as well as introducing active resource-directed demand generation. These extensions relieve some of the user's burden of identifying demands essential for successful mission accomplishment. Finally we showed how our representation is implemented in our prototype system SOSA and how it aids the user in defining the system rules that are used to schedule activities.

This research provides a methodology that enables a system designer to replace the manual, error-prone nature of the current scheduling environment in satellite operations with a decision support system that aids the user. Aid is provided by supporting the definition of scheduling rules (i.e., mission demands, mission types, and command plans), reducing the level of user knowledge required to schedule activities effectively, and automating demands that can be generated based on available data.

Acknowledgements

Support for this research was provided to the second author by Wright State University and by the Ohio Board of Regents through the OBR Research Challenge Fund.

References

Brown, S., & Cox, M. (1999). Planning for Information Visualization in Mixed-Initiative Systems. In M. T. Cox (Ed.), *Proceedings of the 1999 AAAI-99 Workshop on Mixed-Initiative Intelligence* (pp. 2-10). Menlo Park, CA: AAAI Press.

Burstein, M. and McDermott, D (1994). *Mixed-Initiative Military Planning: Directions for Future Research and Development.*

Cesta, A., Oddi, A., & Susi, A. (1999). O-OSCAR: A Flexible Object-Oriented Architecture for Schedule Management in Space Applications. In *Proceedings of the Fifth International Symposium on Artificial Intelligence, Robotics and Automation in Space* (I-SAIRAS-99).

Cox, M. T., & Veloso, M. M. (1997). Controlling for unexpected goals when planning in a mixed-initiative setting. In E. Costa & A. Cardoso (Eds.), *Progress in Artificial Intelligence: Eighth Portuguese Conference on Artificial Intelligence* (pp. 309-318). Berlin: Springer.

Dorn, J., Girsch, M., & Vidakis, N (unpublished). *DEJAVU: A Reusable Framework for the Construction of Intelligent Interactive Schedulers.* http://www.dbai.tuwien.ac.at/proj/DejaVu/document/docu.htm

Ferguson, G., Allen, J. F., & Miller, B. (1996). TRAINS-95: Towards a mixed-initiative planning assistant. In *Proceedings of the Third International Conference on AI Planning Systems.* Edinburgh, Scotland, May 29-31.

Kern, S.C.G. (2000). *Designing a Decision Support System for Satellite Operations Scheduling.* MS thesis, Dept. of Electrical and Computer Engineering, Air Force Institute of Technology.

Lieberman, H. (1999). *Attaching Interface Agents to Applications.* Unpublished.

Oates, T., & Cohen, P. R. (1994). Toward a plan steering agent: Experiments with schedule maintenance. In *Proceedings of the Second International Conference on Planning Systems* (pp. 134-139). Menlo Park, CA: AAAI Press.

Smith, S.F., & Becker, M. (1997) An Ontology for Constructing Scheduling Systems. In *Working Notes of 1997 AAAI Symposium on Ontological Engineering,* Stanford, CA.: AAAI Press.