ConCEPT: Constraint-Checking Editor for Procedure Editing and Tracking

Mark Boddy, Martin Michalowski, Hazel Shackleton
Adventium Labs, Minneapolis MN USA
{firstname.lastname@adventiumlabs.com}

Russell Bonasso, Scott Bell
TRACLabs, Houston TX USA
{bonasso@traclabs.com, scott@traclabs.com}

Abstract

Constructing, maintaining, modifying, and adapting operational procedures for manned space operations is a complex task. The procedure author is required to keep track of state constraints such as the location of personnel, equipment, or tools, and of resources such as oxygen, fuel, or power. They must also keep in mind a set of constraints imposing additional restrictions on these procedures. For operations on the International Space Station (ISS), these constraints may be of several different types, including such things as warnings that must be present for a given type of operation, previous actions that must have been taken, tracking the location of personnel, tools, and equipment, or synchronizing operations by different astronauts.

As part of an ongoing research project funded by NASA, Adventium Labs and TRACLabs have designed and implemented an initial version of the Constraint Checking Editor for Procedure Tracking (ConCEPT) system, a constraint-checking system for procedures represented in the Procedure Representation Language (PRL). ConCEPT has been integrated into TRACLabs' Procedure Integrated Development Environment (PrIDE), so that procedures in PRL can be checked against constraints and modified during the process of procedure authoring. The design of ConCEPT, including the types of constraints considered and the integration into the PrIDE user interface, has been validated in discussions with NASA flight controllers.

1 Introduction

NASA's manned space operations, for example those involving the ISS, rely heavily on an extensive database of *procedures*, which specify what steps are to be taken, in what order, and what set of confirmation checks must be done as the operation proceeds. Here is a simple procedure specifying how a single crew member moves from the open airlock to outside of the ISS:

```
Procedure (tether)
1. Thermal cover - open
2. Egress AIRLOCK
3. Attach tether to left D-ring extender
4. Verify tether config
```


More complex procedures may be *hierarchical*, in the sense that one step of a given procedure may refer to another procedure (or, in some cases, to a step or steps contained in another procedure). Procedures for more complex operations may require tracking resources such as oxygen, fuel, or tools; choosing from alternative sub-procedures; synchronized or overlapping action by multiple agents; and preconditions required for the procedure to be executed. Figure 1 shows an example of a more complex, hierarchical procedure drawn from the same Extra-Vehicular Activity (EVA) domain as the simple example above.

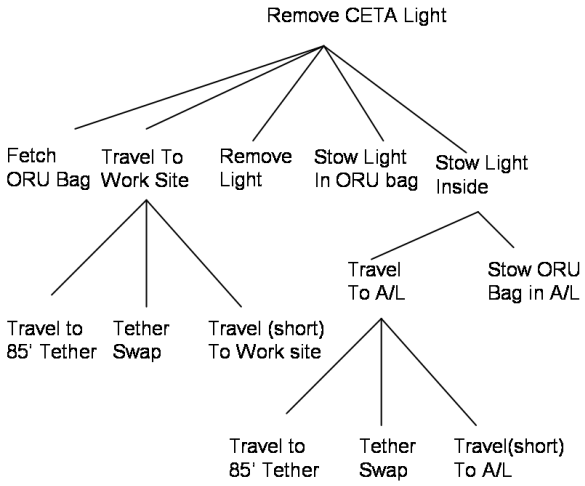

Figure 1: EVA procedure with hierarchical structure.

Further complicating the picture is the presence of additional constraints on the procedure or on the current state as the procedure is executed. Examples of these procedural constraints include never operating outside the ISS without being tethered, or keeping an external light heated when it is turned off in order to extend its service lifetime.

Over the past few years, several projects have aimed to provide NASA flight controllers with operations planning and execution aids, most notably the Automation for Operations initiative (Frank 2009). One result of this work has been the development of the Procedure Representation Lan-

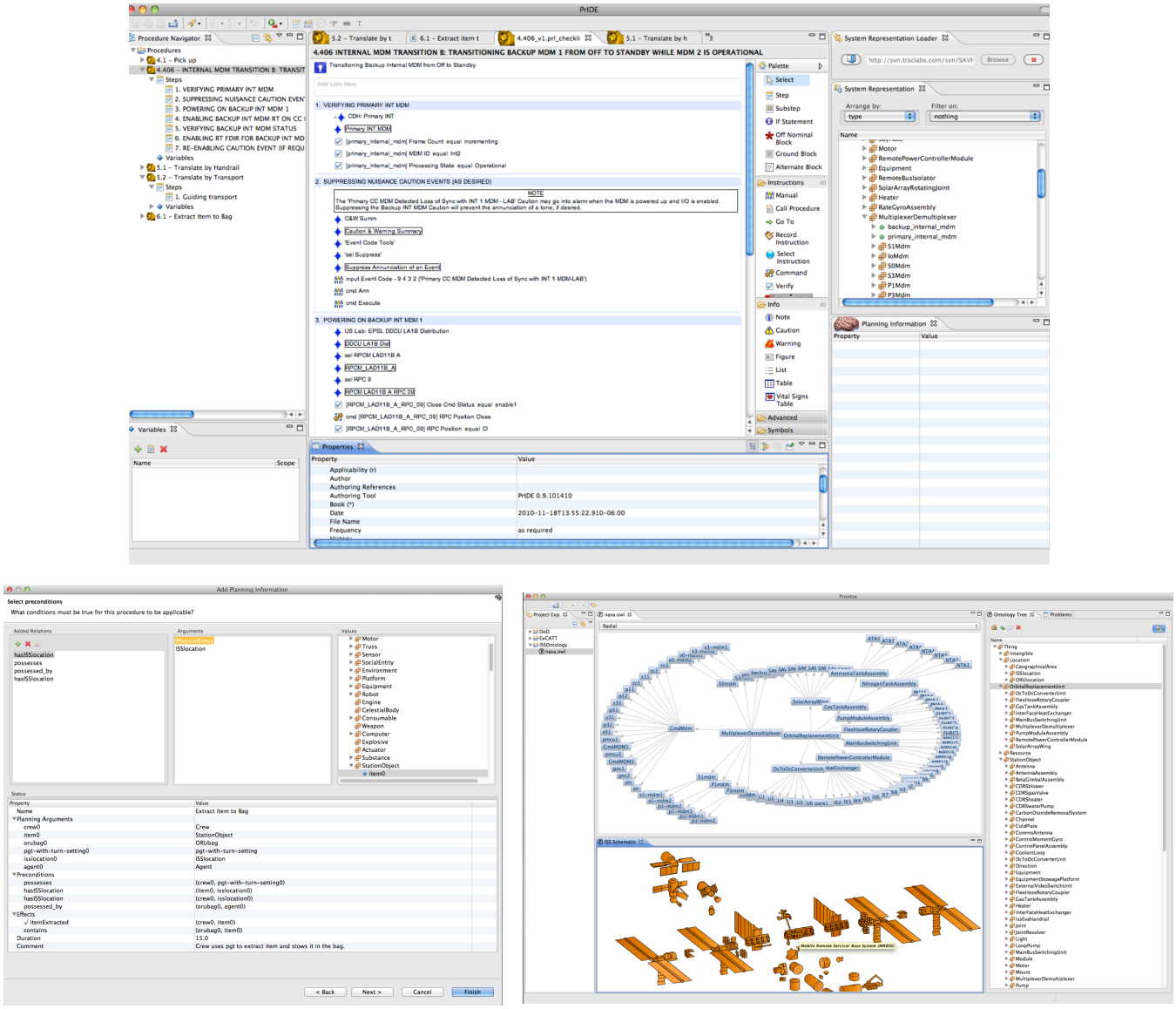

Figure 2: The Procedure Integrated Development Environment (PrIDE), the PrIDE Planning Wizard (PPW) and the PrIDE Ontology Editor (PRONTOE), at top, bottom left and right, respectively

guage (PRL), an XML-based language developed by TRACLabs that captures the form of traditional procedures and automatically translates them into code that can be executed by NASA-developed autonomous executives. PRL provides for access to spacecraft and habitat telemetry, includes constructs for human-centered displays, allows for the full range of human interaction, and is intended to support automatic methods of verification and validation. PRL is supported by a graphical authoring system, known as PrIDE, that enables non-computer programmers to write procedures (Kortenkamp, Bonasso, and Schreckenghost 2007).

PrIDE is intended to replace processes in which procedure construction, maintenance, adaptation, and monitoring during execution have been done without the aid of tools more sophisticated than a word processor. For example, many procedures for ISS operations are authored in Microsoft Word, in some cases resulting in documents that are dozens of pages long, with extensive cross-referencing both within and among documents.

Over the past few years, Adventium Labs has teamed with TRACLabs on a series of development projects that have added new capabilities and support to PrIDE for creating and editing PRL procedures. One of these projects resulted in the implementation of the PrIDE Planning Wizard (PPW), in which the PRL schema were extended to provide explicit support for Hierarchical Task Network (HTN) planning, which can then be used to manage hierarchical procedures structures such as that shown in Figure 1 (Bonasso,

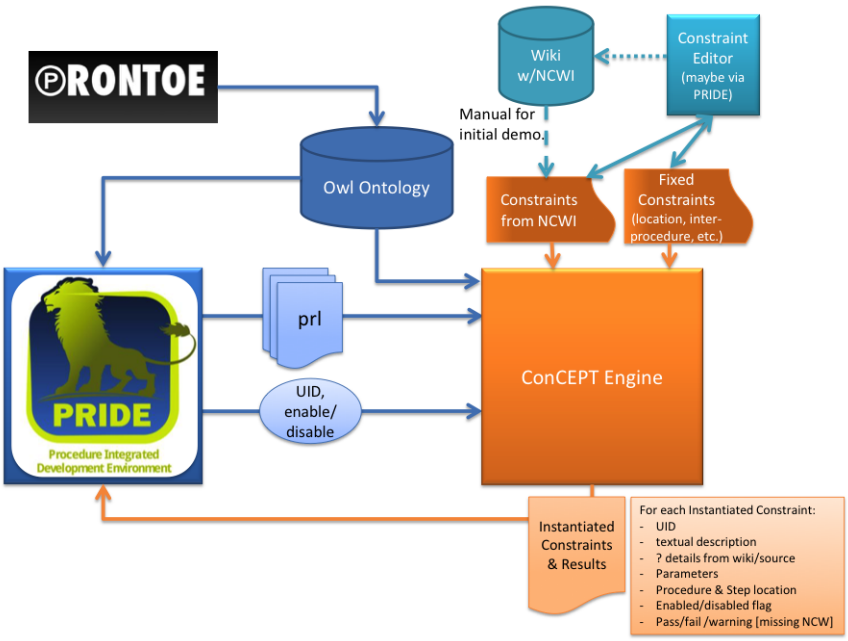

Figure 3: ConCEPT system architecture.

Boddy, and Kortenkamp 2009; Boddy and Bonasso 2010; Bonasso and Boddy 2010). PPW supports users who are not experts at developing and maintaining planning models by providing a "base ontology" from which domain objects can be specialized.

A second joint project focused on the domain model. Planning in any moderately complex domain depends crucially on being able to maintain a model of the environment and the current state. The PrIDE Ontology Editor (PRONTOE) provides explicit support for non-expert users (that is, non-expert in the maintenance of planning domain models) to maintain and extend an ontology of object types, along with providing interfaces to external systems such as NASA's External Configuration Analysis and Tracking Tool (ExCATT), making it easier to update current state information (Bonasso et al. 2013; Bell et al. 2013). Figure 2 shows the main display panels for PrIDE, PPW, and PRONTOE.

The work described in this paper is a logical extension of our previous collaborations. The Constraint Checking Editor for Procedure Tracking (ConCEPT) provides automated constraint checking related to state, resources, and procedural constraints for flight controllers creating or editing procedures. Integrated with PrIDE, ConCEPT can be used to check constraints on procedures as they are created or edited, thus reducing errors, rework, and hand debugging of procedures as they are being constructed. In the rest of this paper, we describe the current ConCEPT implementation (Section 2), followed by a detailed description of the kinds of constraints currently supported by ConCEPT (Section 3). We then briefly describe and motivate the technical choices we have made in implementing ConCEPT, and discuss plans for further development over the remainder of the current project and beyond (Section 5).

2 ConCEPT Implementation and Integration

Figure 3 shows how ConCEPT is integrated functionally into PrIDE. A human user creates or edits procedures using PrIDE, occasionally invoking ConCEPT to check for violations of several different classes of constraints on those procedures. The results of those checks are returned to PrIDE to be displayed to the user, who may choose to fix, delete, or simply ignore the reported constraint violations. As shown in the figure, ConCEPT builds on our previous work, making use of the domain model developed and maintained using PRONTOE. Also shown is the path by which some constraints are imported into ConCEPT, in this case from a wiki containing information on Notes, Cautions, Warnings, and Inhibits that must be present for EVA operations on the ISS. This import process and the accompanying "Constraint Editor" are the only parts of the architecture that are not fully implemented in the current version of the system.

ConCEPT has been implemented and integrated into PrIDE. Figure 4 shows the running system. Through the rest of the paper, all of the screenshots depicting ConCEPT functions are taken from the running system, displayed either as part of the display in Figure 4, or as popups on it.

3 Constraint Examples

Based on our discussions with EVA personnel, our initial implementation of ConCEPT includes four general classes of constraints, specialized to fit more closely with EVA operations and procedures.

Notes, Cautions, and Warnings constraints enforce the presence of applicable Notes, Cautions, or Warnings in the appropriate place in a procedure. The more generic

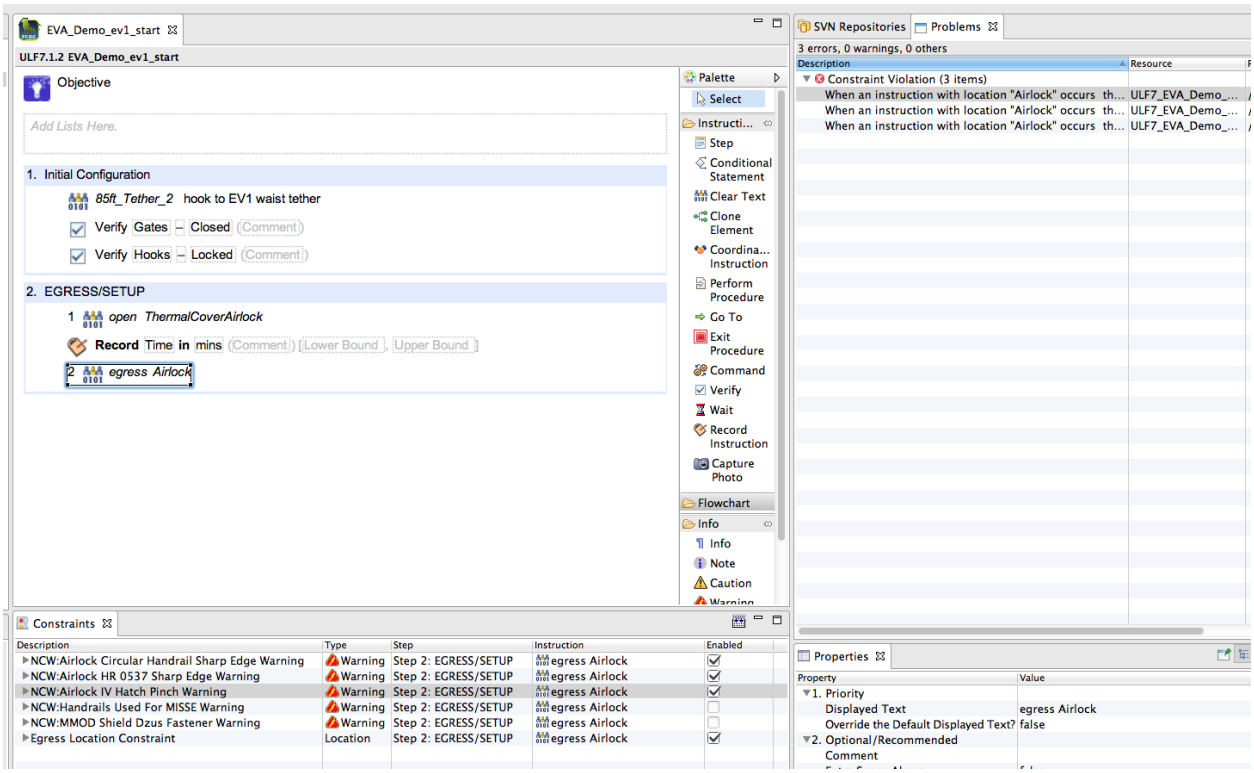

Figure 4: PrIDE display augmented with ConCEPT, showing a missing Pinch Warning (highlighted in bottom left pane) for Egress step highlighted in the upper right pane

form of this constraint is a check for specified text to be displayed, along with a description where to display it.

Location constraints refer to the projected locations of both crew and objects such as tools. The more generic form of this constraint is state progression: define what actions change which aspects of state, and then track those changes over time.

Operational constraints refer to collections of procedure steps or instructions, for example ensuring that if some actions are present in a given procedure or set of procedures, so are a specified set of others.

Synchronization constraints enforce required synchronization across procedures, for example for Give_Go / On_Go operations, or for operations that require simultaneous actions by multiple agents.

These constraint types are described in more detail, below.

3.1 Notes, Cautions, and Warnings

These constraints enforce the presence of Notes, Cautions, and Warnings for relevant procedure instructions. For example, Figure 4 shows a missing "Pinch Warning" for an Egress step in the procedure shown. The relevant constraint, shown in Figure 5, has been imported from the wiki entry shown in Figure 6. When the constraint violation shown in the lower pane is selected, ConCEPT generates a proposed "quick fix," which in this case is simply to add the required Warning, resulting in the procedure displayed in Figure 7.

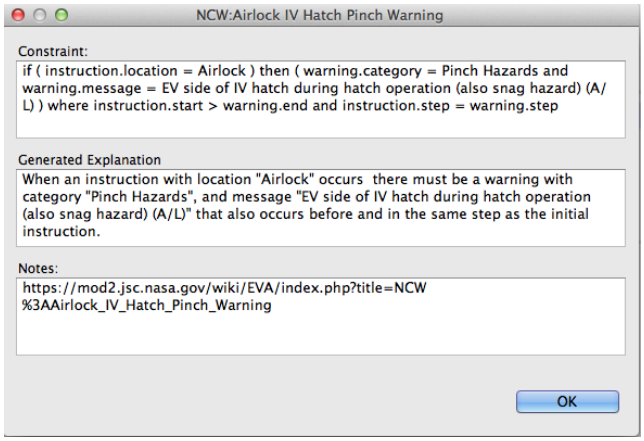

Figure 5: Relevant "Pinch Warning" constraint

3.2 Location

Location constraints track the projected locations of both crew and objects such as tools. For example, in Figure 8, the instruction outlined in blue specifies that certain tools should be stowed in crewlock bag #2. But in a previous step also shown, that bag was left at the airlock immediately after Egress, after which the crew member translated to a different location.

As shown in Figure 9, a popup in the PrIDE user in-

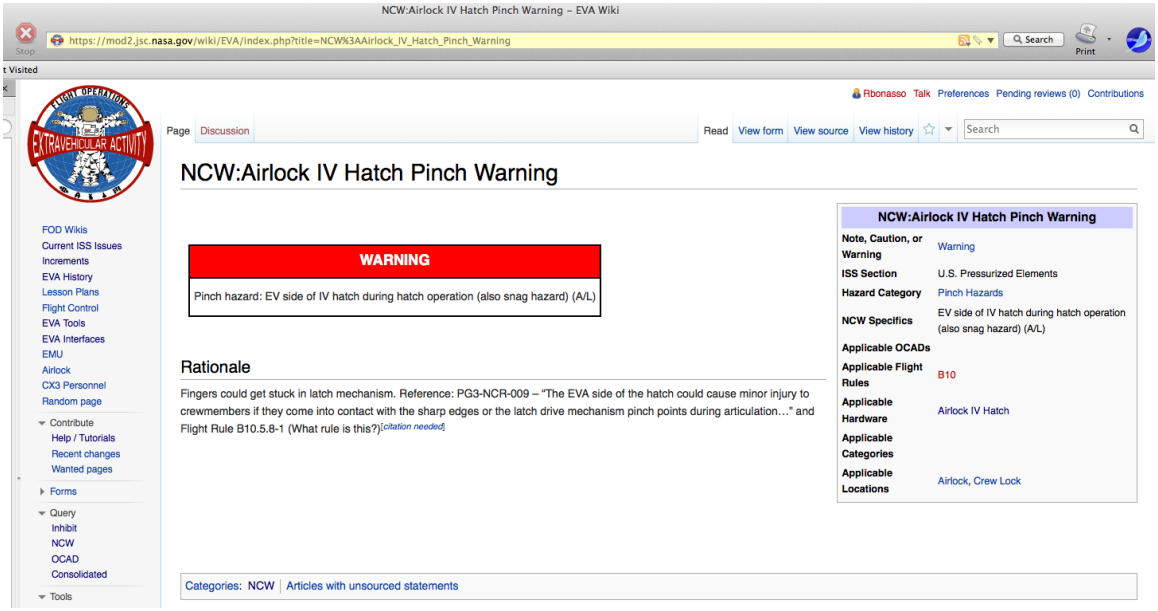

Figure 6: Wiki source for Pinch Warning

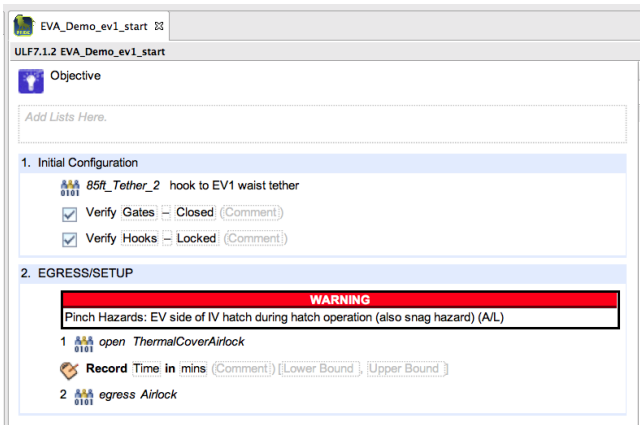

Figure 7: Fix for Pinch Warning has been applied

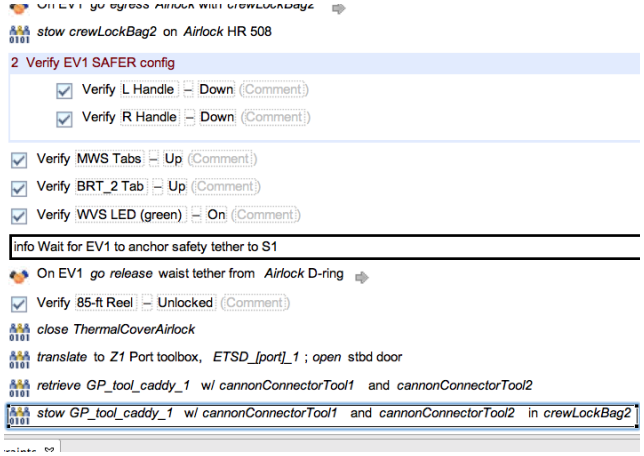

Figure 8: Crewlock bag #2 was left stowed at the airlock.

terface warns that crewlock bag #2 will not be at the required location for the highlighted instruction. The popup also specifies the last location known for the item. For Location constraints, no "quick fix" is suggested, because it is not clear what that fix should be. In the example given, should the author add a step to retrieve the bag before translating away? Or should some crew member go get the bag later? Or should the tools be stowed elsewhere?

The relevant constraint template in this case is:

```
at(instr,procedure.agent.location = instr
    .target.location);
```

This constraint says that when the specified instruction starts, the agent and any object (target) mentioned in the instruction (the bag, in this case) must be at the same location. If the instruction or the containing step specifies a location, then we could add an additional constraint, which is that both the crew and the bag are at the specified location:

```
at(instr, procedure.agent.location =
    instr.location);
at(instr, instr.target.location = instr.
    location);
```

This would have the added benefit of allowing us to determine independently whether the agent or the bag is at the wrong location.

Checking and enforcing location constraints requires the ability to track location through the execution of the procedure. Procedure steps are ordered within a single procedure and we can specify the instruction types which change the location of an agent or object, so this is straightforward. It

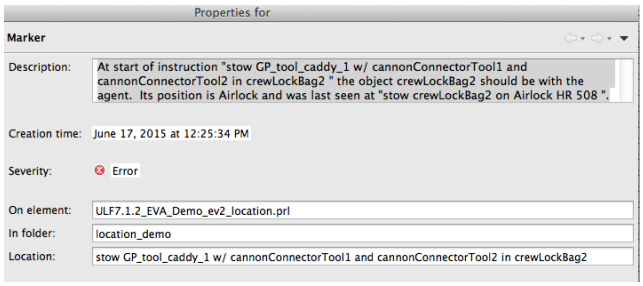

Figure 9: ConCEPT provides the most recent known location for the bag.

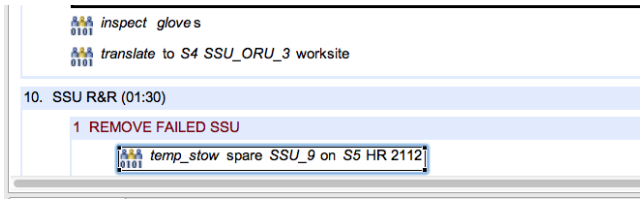

Figure 10: EV1 is at S4, the "stow" instruction specifies S5.

becomes more complicated when the context includes more than one procedure, as in the example in Section 3.4.

This inference is a simple form of *progression*, as in AI planning, with some added axioms to make sure that objects attached to the crew member, and anything contained in those objects, will change location as well. As discussed in Section 5, a more general planning capability may eventually be needed, but has not yet been included in ConCEPT. Location constraints can be known to be violated—the object is in the wrong place, according to the procedure(s) as written—known to be satisfied, or unknown, for example if there are multiple procedures and insufficient coordinating instructions to establish an ordering on the relevant procedure steps.

The situation shown in Figure 10 illustrates one of the difficulties in modeling the ISS as a planning domain: "location" on the ISS is a complicated concept. In this example, the crew member moves to the specified location S4, then stows a piece of equipment on a hand-rail at S5. ConCEPT generates an error message:

> At start of instruction "temp_stow spare SSU_9 on S5 HR 2112" the agent should be at S5. Their position is S4 and was last seen at "translate to SS4 SSU_ORU_3 worksite."

What the system does not recognize is that being positioned at S4 *is* the correct location for working on that equipment. For more details on the complexities involved in modeling and reasoning about ISS locations, see (Bell et al. 2013).

3.3 Operational Constraints

Operational constraints ensure that if some actions are present in a given procedure or set of procedures, so are a specified set of others. For example, before any work on or

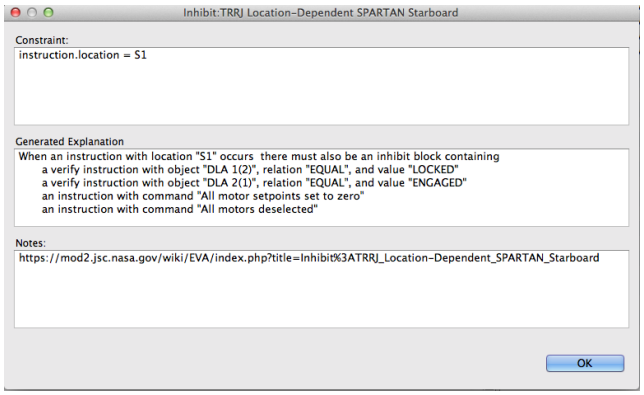

Figure 11: Applicable constraint for missing inhibits

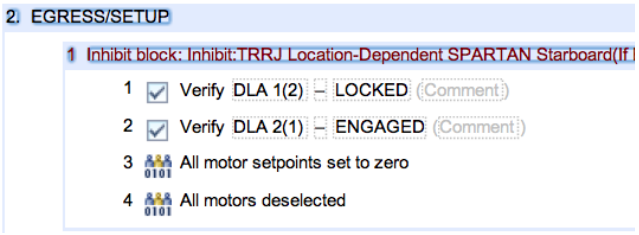

Figure 12: Applied Fix for Missing Inhibits

near a Solar Array Alpha Rotary Joint (SARJ), the SARJ itself must be locked down. From the EVA wiki:

> If EV crew working within 2 feet, outboard of Starboard SARJ, or required per loads FR (includes outboard faces of outboard ELCs/ESPs)
>
> ✓ DLA 1(2) — LOCKED
> ✓ DLA 2(1) — ENGAGED
> All motor setpoints set to zero
> All motors deselected

The four items listed are special checks, known as *inhibits*, that must occur in a block at the top of the appropriate procedure step. Figure 11 shows the applicable constraint. The quick fix in this case is to add an "inhibit block" at the beginning of the step, with the result shown in Figure 12.

In Figure 13, we see another example of an operational constraint. In this case, a bolt ("BAX bolt #2") is being installed. For this installation, there must also be explicit instructions for recording torque and turns for that installation. Slightly more formally:

> For any instruction, i, that involves installing an object of type BOLT, there must be two additional instructions i1 and i2 in the same step as i and after i, to record torque and turns information for the installation of the bolt.

The procedure in Figure 13 satisfies this constraint, as the installation step does have the required reporting instructions.

Figure 13: BAX bolt installation with required recording instructions

3.4 Synchronization

EVA operations involve several different kinds of synchronization among the EVA crew, between EVA and IV (crew within the ISS participating in the operation in some way), or between crew and flight controllers on the ground. To support this synchronization, PRL and PrIDE have recently been extended to support "coordinated instructions" across multiple procedures. ConCEPT includes a class of *synchronization constraints*, which can be used to enforce the presence of coordinated instructions, including the presence of the associated instructions in other procedures.

Figure 14 shows the modified PrIDE interface with two procedures side-by-side. The highlighted instructions in each procedure are coordinated, such that EV2 is instructed to wait for EV1 to provide a "go" to proceed with further operations. It is a constraint violation for either "give go" or "on go" instructions to appear without an explicit pointer to the complementary instruction in some other procedure. Coordinated instructions can be used to infer ordering on steps and instructions *across* procedures, thus increasing the power of the system to reason about state changes such as object location, in a multi-agent environment.

4 Technical Details

To this point in the paper, we have presented a lot of examples and screenshots, but little in the way of description of how the underlying inference is accomplished. That emphasis matches how the project has been conducted so far. ConCEPT is strongly driven by user interaction and workflow, as well as by the information available through integration to NASA information systems or through the other PrIDE extensions discussed above.

Initially, our design for ConCEPT called for the integration of a fully functional Constraint Satisfaction Problem (CSP) solver, as well as the use of the Action Notation Modeling Language (ANML) for representing procedural information, supporting a limited planning capability encoded in the underlying CSP (Smith, Frank, and Cushing 2008). In practice, neither of these things has been necessary. While ConCEPT is solving a CSP, to date the constraints involved have been simple enough, few enough, and sufficiently independent from one another that a general purpose solver provides much more power than is needed, to the point of not being worth the additional overhead.

Similarly, the inference being performed in tracking the location of both personnel and objects such as tools will quickly be recognized by anyone remotely familiar with AI planning as an elementary form of state progression. The same technique could be applied to track other forms of state information, but so far that has not been of interest to potential users. Nor does ConCEPT yet exploit the much richer planning information available through the PRL extensions made as part of the PPW, because there has not yet been a strong call from users to implement those extensions, despite their interest in using them, in principle.

Our emphasis has been on getting a working prototype in front of potential users as quickly as possible, so as to maximize the feedback obtained regarding what features and capabilities are or are not of interest. This strategy has proved sound: our initial understanding of the form of constraints that would be most useful was considerably revised over the course of several meetings with EVA flight controllers. At the most recent review, we were assured that what we had implemented, and have presented in this paper, is what they would like to use.

The inferential machinery underlying ConCEPT will almost certainly evolve in complexity and sophistication over time. But our current approach means that those elaborations will be done in the service of specific requirements.

5 Discussion and Future Work

This paper describes the state of the ConCEPT system at roughly the mid-point of a 2-year development project. The focus to date has been primarily on user requirements, interface, and workflow, with implementation of inferential machinery as required. ConCEPT has been demonstrated to and reviewed by potential users, but is not sufficiently robust for a field test. The next step in development will be to shadow the construction of an EVA procedure. In fact, this procedure will be constructed in 3 different ways: NASA personnel will use the existing process (i.e., in Microsoft Word), with other personnel shadowing that process using the currently-available version of PrIDE. In addition, members of the ConCEPT team will also shadow the process, using a newer version of PrIDE augmented with ConCEPT. We expect this exercise to be a rich source of corrections and further requirements, to be addressed over the remainder of the second year of the project.

Looking out past the end of the current project, there are many directions in which this work can be extended. Improvements in robustness and efficiency will be required before the system is deployable. Given the data-intensive nature of ConCEPT's inference, integrating more closely with NASA's existing information systems will be critical to user acceptance, as will providing effective editors for ConCEPT's internal knowledge base, tuned to user workflows. We have extensive previous experience with

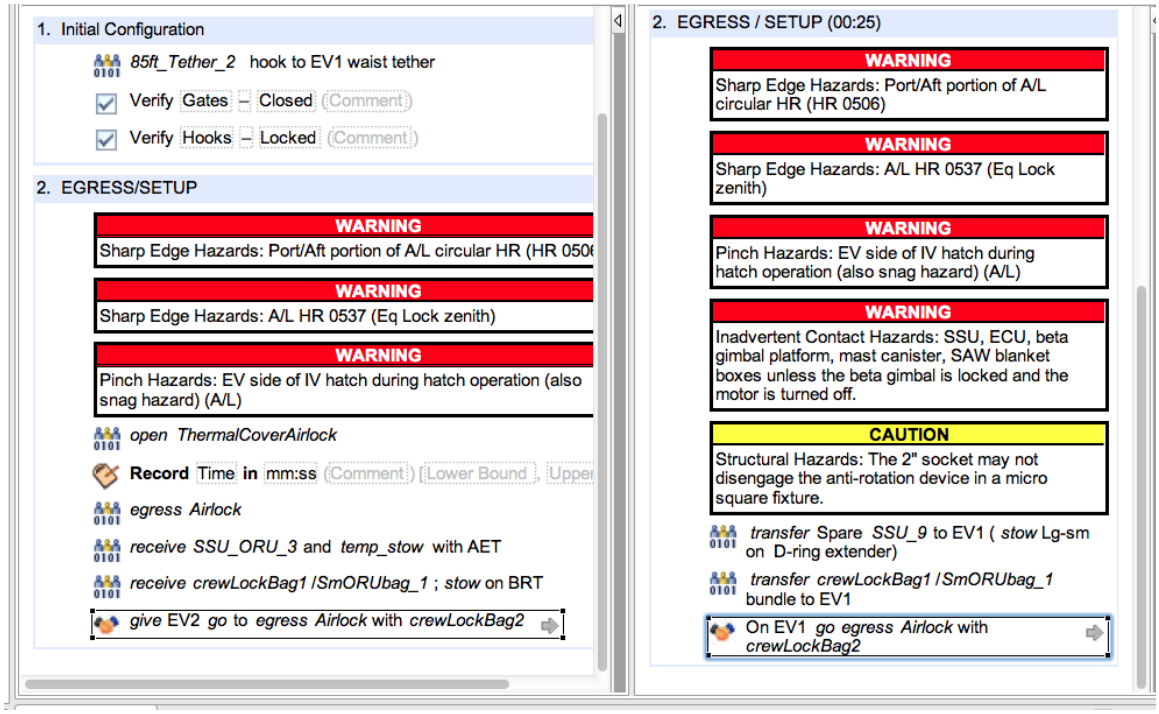

Figure 14: Coordinated instructions have been linked

these issues, as they have arisen with regard to PPW and PRONTOE, as well.

One possible near-term extension would be to apply ConCEPT to EVA procedure execution and on-the-fly procedure modification. Additional potential applications for ConCEPT include flight disciplines beyond EVA, other NASA applications, and other applications where PRL and PrIDE are gaining interest and acceptance, such as the oil and gas industry.

Longer term, there are some significant research questions to be addressed. Use of the full power of the PPW planning extensions will require more systematic treatment of time and state. Temporal durations and metric state information are likely to become important, as the breadth of application grows. On the CSP side, presenting a single violated constraint to the user as an explanation suffices for now, because interaction among those constraints is minimal. But as the complexity, diversity and number of constraints grows, implementing, or more likely adopting, a CSP engine that is capable of identifying and presenting more complex inconsistencies will become increasingly important.

Acknowledgements: This material is based upon work supported by NASA via the NASA Shared Services Center, Stennis Space Center, MS under Contract No. NNX14CA10C. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of NASA Shared Services Center, Stennis Space Center, MS.

References

[Barreriro and Chachere 2010] Barreriro, J. J., and Chachere, J. 2010. Constraint and flight rule management for space mission operations. In *International Symposium on Artificial Intelligence, Robotics, and Automation in Space*.

[Bell et al. 2013] Bell, S.; Bonasso, R.; Boddy, M.; Kortenkamp, D.; and Schreckenghost, D. 2013. Prontoe: A case study for developing ontologies for operations. In *5th International Conference on Knowledge Engineering and Ontology Development*.

[Boddy and Bonasso 2010] Boddy, M., and Bonasso, R. 2010. Planning for human execution of procedures using anml. In *ICAPS 2010 Scheduling and Planning Applications Workshop*.

[Bonasso and Boddy 2010] Bonasso, P., and Boddy, M. 2010. Eliciting planning information from subject matter experts. In *ICAPS 2010 Workshop on Knowledge Engineering for Planning and Scheduling (KEPS)*.

[Bonasso et al. 2013] Bonasso, R. P.; Boddy, M.; Kortenkamp, D.; and Bell, S. 2013. Ontological models to support space operations,. In *Workshop on AI in Space at the 2013 International Joint Conferences on Artificial Intelligence (IJCAI)*.

[Bonasso, Boddy, and Kortenkamp 2009] Bonasso, P.; Boddy, M.; and Kortenkamp, D. 2009. Enhancing nasa's procedure representation language to support planning operations. In *Proceedings of the International Workshop on Planning and Scheduling for Space (IWPSS)*.

[Frank 2009] Frank, J. 2009. Automation for operations. `http://ti.arc.nasa.gov/news/a4o-demo-for-hdu/`.

[Kortenkamp, Bonasso, and Schreckenghost 2007] Kortenkamp, D.; Bonasso, R.; and Schreckenghost, D. 2007. Developing and executing goal-based, adjustably autonomous procedures. In *AIAA InfoTech@Aerospace Conference*.

[Smith, Frank, and Cushing 2008] Smith, D.; Frank, J.; and Cushing, W. 2008. The anml language. In *International Conference on Automated Planning and Scheduling*.